VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Měření hmotnosti pomocí senzorů připojených k Raspberry Pi

Weight Measument using Sensors Connected to Raspberry Pi

Lukáš Štěpán

Bakalářská práce

Vedoucí práce: Ing. Pavel Nevlud

Ostrava, 2021

Abstrakt

Cílem práce je návrh automatizovaného měření hmotnosti s využitím váhového senzoru a zobrazení naměřených dat v grafech na webových stránkách. Nejprve jsou popsány použité hardwarové a softwarové prostředky. Práce se dále zabývá zajištěním měření a ukládání dat do databáze, a následnou tvorbou webové aplikace pro jejich zobrazení s použitím knihovny Dash. Nakonec je popsáno nastavení webového serveru pro aplikaci na zařízení Raspberry Pi.

Klíčová slova

Dash, Raspberry Pi, tvorba grafů, váhový senzor, webová aplikace, webový server

Abstract

The purpose of this thesis is to design an automated weight measurement with the use of a weight sensor and display the data in graphs on a web page. First, the hardware and software resources used are described. Next, the work deals with obtaining and storing the data in a database, followed by creating a web application for plotting the data with the use of the Dash library. Lastly, the work describes setting up a web server for the application on a Raspberry Pi device.

Keywords

Dash, graph plotting, Raspberry Pi, web application, web server, weight sensor

Poděkování

Rád bych na tomto místě poděkoval vedoucímu práce Ing. Pavlovi Nevludovi a všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Obsah

Seznam použitých zkratek a symbolů

ADC	–	Analog-to-Digital Converter
ASCII	–	American Standard Code for Information Interchange
CSS	–	Cascading Style Sheets
CSV	–	Comma-separated values
GPIO	–	General-purpose input/output
HDMI	–	High-Definition Multimedia Interface
HTML	–	Hyper Text Markup Language
HTTP	–	Hypertext Transfer Protocol
I2C	–	Inter-Integrated-Circuit
IoT	–	Internet of Things
IP	–	Internet Protocol
NAT	–	Network address translation
OS	–	Operating System
PGA	–	Programmable-gain amplifier
SD	–	Secure Digital
SPI	–	Serial Peripheral Interface
SPS	–	Samples per Second
SQL	–	Structured Query Language
SSH	–	Secure Shell
TCP	–	Transmission Control Protocol
UART	–	Universal Asynchronous Reception and Transmission
USB	–	Universal Serial Bus
WSGI	–	Web Server Gateway Interface

Kapitola 1

Úvod

Cílem této práce je zařídit automatizované měření hmotnosti s použitím váhového senzoru. Naměřená data poté ukládat do databáze a zobrazovat vykreslená do grafů na webových stránkách.

V práci jsou nejprve popsány hlavní protředky použité k realizaci úkolů, jako hmotnostní senzor, AD převodník, programovatelná deska Arduino, použitý model mikropočítače Raspberry Pi a využitý software.

Dále je popsáno získání samotných hmotnostních dat, k tomu je použit tenzometrický hmotnostní senzor připojený k A/D převodníku s obvodem HX711 převádějícímu a zesilujícímu signál senzoru. Signál senzoru je zpracován a kalibrován pomocí vývojové desky Arduino Uno, která vypisuje potřebná data do USB sériového monitoru.

Naměřená data se dále ukládají do databázového souboru skriptem napsaným v programovacím jazyku Python s využitím knihoven sqlite3 pro přístup do databáze, a pyserial, umožňující odečítat hodnoty na sériovém portu.

Dalším krokem je vytvoření webové aplikace pro tvorbu grafů. Aplikace je taktéž napsána ve skriptovacím jazyku Python a pro konstrukci grafu využívá knihovny Plotly Dash. Program spouští server na příslušné adrese a vytváří HTML grafy podle vlastního nastavení tak, aby byly přístupné na webovém prohlížeči na lokální síti.

Aby bylo možné přistupovat k aplikaci i z internetu, musí běžet na webovém serveru. Server běží na zařízení typu Raspberry Pi a je realizován službou Nginx, chovající se jako reverzní proxy, a službou Gunicorn, která slouží jako rozhraní mezi webovým serverem a webovou aplikací.

Nakonec je pro přístup z internetu nutné nastavit přesměrování portu a pravidla firewallu na internetovém routeru v síti, kde se webový server provozuje. To zařizuje volný přístup k aplikaci pomocí IP adresy a portu serveru.

Kapitola 2

Popis použitých prostředků

2.1 Hmotnostní senzor

Hmotnostní senzory (load cell) jsou převodníky konvertující fyzickou zátěž jako napínání, tlak, kompresi, točivý moment, apod. na elektrický signál, který může být následně zmeřen a zpracován. Hmotnosní senzory se podle jejich použití dělí na několik typů: [1]

Jednobodové snímače – pro použití u zařízení s malou až střední velikostí. Například stolní váhy a plnící či balící stroje.

Nízkoprofilové ohybové snímače – používané v případech, kde je pracovní místo omezené, a je potřeba malého měřícího zařízení. Například poštovní, obchodní či zdravotnické váhy. Obvykle jsou používané jako skupina tří nebo čtyř snímačů.

Nosníkové ohybové snímače – pro použití v paletových a dopravních vahách, zásobnících či menších násypkách.

Dvojité nosníkové snímače – používané například u nákladních vozidel, nádržích a násypkách.

Tahové snímače typu S – pro tažné použití většinou jako závěsné váhy, například u jeřábů nebo váh sloužících k měření velkých pytlů či kontejnerů.

Kompresní snímače – používané ve velkých vahách nebo vážících mostech na nákladní vozidla.

Válcové a kruhové snímače – používané k vysoce přesnému vážení zásobníků, nádrží, násypek, sil a kolejových vozidel.

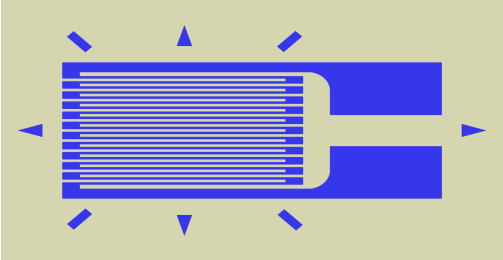
Palubní snímače – pro palubní vážící systémy nákladních vozidel, traktorů a dalších vozidel.

Kolíkové snímače (Loadpins) – používané k měření různých dynamických, statických a zvedacích sil.

Vážící podložky (Weighpads) – používané jako přenosné váhy aut, nákladních a dalších vozidel.

2.1.1 Tenzometry

Nejčastěji používané hmotnostní senzory jsou senzory tenzometrické jejichž základní funkčí součástkou je tenzometr. Tenzometry jsou pasivní čidla, nalepená nebo připevněná na deformovatelné těleso převádějící deformaci na změnu elektrického odporu tenzometru. Tenzometrické senzory jsou tedy kovová tělesa, většinou z hliníku či oceli, na která bylo připevněno několik tenzometrů. Při zátěži působící na těleso váhového senzoru se těleso lehce deformuje a pokud není přetíženo, je schopné se opět vrátit do původního stavu. Deformaci zaznamenají tenzometry senzoru a změní svůj měrný odpor přímo úměrně aplikovanému zatížení.[2]

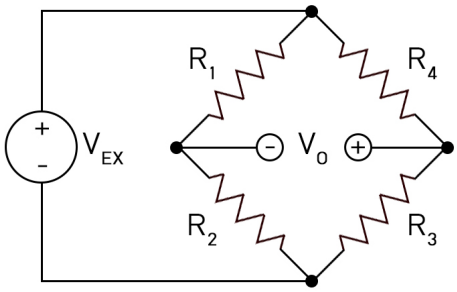


Obrázek 2.1: Fóliový tenzometr [3]

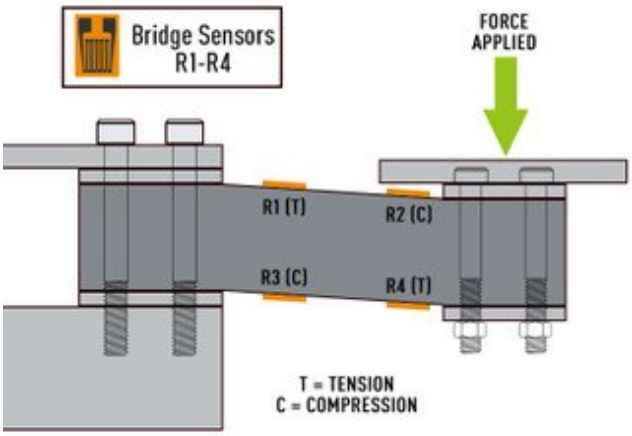
2.1.2 Princip tenzometrického senzoru

Měření funguje na principu Wheatstonova můstku, což je obvod skládající se ze dvou větví po dvou odporech nebo tenzometrech připojených na stejnosměrný zdroj. Větve fungují jako děliče napětí. Pokud jsou odpory vyrovnané, tak výstupní napětí V0 je nulové. Pokud je však tenzometr deformován a tím změní svůj odpor, tak je napětí změněno.[2]

$$V_0 = (\frac{R_3}{R_3 + R_4} - \frac{R_2}{R_1 + R_2}) \cdot V_{EX} \tag{2.1}$$



Obrázek 2.2: Wheatstonův můstek [4]



Obrázek 2.3: Princip tenzomezrického senzoru [5]

2.1.3 Fóliové tenzometry

Samotný tenzometr je tvořen elektrickým vodičem, většinou velmi tenkým drátkem či fólií, uspořádaným do mřížkového vzoru. Deformací tohoto vzoru dochází ke změně odporu tenzometru. Fóliové tenzometry se také podle jejich využití dělí na několik typů jako například: [6]

Lineární – obsahují jednu měřící mřížku pro měření zátěže pouze v jednom směru.

Dvojité lineární – tvořené dvěmi, paralelně uspořádanými měřícími mřížkami. Používané typicky pro měření na nosnících.

Smykové – tvořené dvěmi mřížkami uspořádanými do tvaru V. Typickou aplikací je měření na torzních tyčích a měření smykových napětí.

Poloviční můstkové – používají dvou měřících mřížek připevněných na měřené těleso tak, aby se jejich signály sčítaly. Zbylé dva odpory můstku mají pevnou hodnotu. Používané typicky pro měření ohybu nosníku.

Plné můstkové – využívající čtyř měřících mřížek pro měření nosníků s vyšší přesností.

Ružicové tenzometry – tvořené třemi měřícími mřížkami uspořádanými v úhlech 0°/45°/90° nebo 0°/60°/120°. Používané pro měření deformace ve více směrech.

2.1.4 Další typy hmotnostních senzorů

Mimo tenzometrických senzorů existují i senzory pracující na jiném principu. Mezi ně patří například:[6][7][8]

Hydraulické – při jejich zatížení se stlačuje plnící tekunina uzavřená v elastomerní membránové komoře. Zvyšováním působící síly stoupá tlak kapaliny, který je následně měřen.

Pneumatické – fungují na principu vyvážených sil. Měří změnu tlaku působící na komory tlumičů.

Piezoelektrické (polovodičové) – fungují podobně jako tenzometrické senzory. Používají piezoelektrických snímačů, které mění svůj elektrický odpor deformací polovodičového krystalu.

Magnetostrikční – pracuje na principu změny permability feromagnetického materiálu při aplikaci síly.

Strunové – jsou senzory využívají kmitající struny. Frekvence kmitání se mění v závislosti na deformaci struny při zatížení.

2.1.5 Použitý senzor

Pro práci byl použit hmotnostní senzor s označením **CZL601** s maximální nosností do 100 Kg. Senzor je vyroben ze slitiny hliníku a je hermeticky utěsněn lepidlem podle stupně krytí IP65. Je určen pro kuchyňské či podlahové váhy a další zařízení. [9]

Obrázek 2.4: Hmotnostní senzor CZL601 [10]

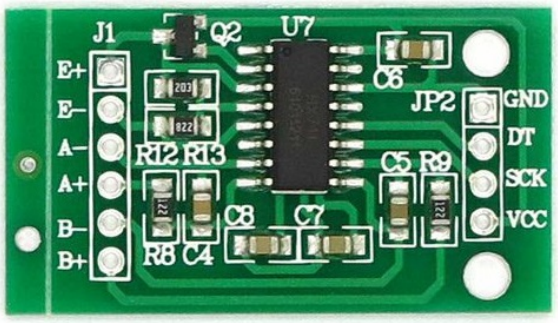
Tabulka 2.1: Vodiče senzoru CZL601 [9]

Barva vodiče	**Funkce vodiče**
Červená	Excitace (+)
Černá	Excitace (-)
Zelená	Výstup (+)
Bílá	Výstup (-)

2.2 Modul pro váhové senzory s obvodem HX711

Modul s obvodem HX711 je AD převodník s 24-bitovou přesností, designovaný pro váhové senzory. Obvod HX711 obsahuje dva vstupní kanály A a B s programovatelným zesílením (PGA). Kanál A může být naprogramován na zisk 128 nebo 64. Kanál B má pevně nastavený zisk 32. Modul disponuje nastavitelným vzorkováním 10SPS nebo 80SPS. [11]

2.2.1 Popis pinů ADC modulu

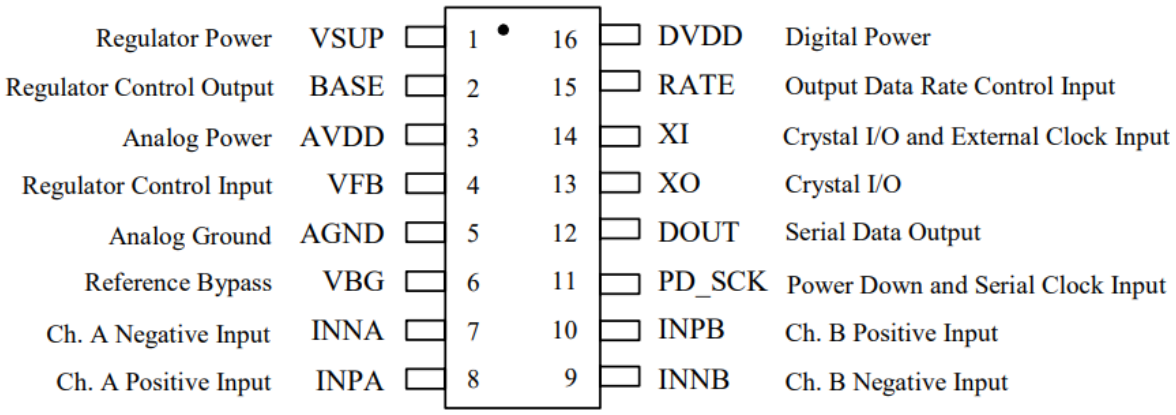


Obrázek 2.5: Modul s obvodem HX711 [11]

Tabulka 2.2: Popis pinů ADC modulu [11]

Značení	**Funkce**
E+	Excitace senzoru (+)
E-	Excitace senzoru (-)
A-	Analogový vstup kanálu A (-)
A+	Analogový vstup kanálu A (+)
B-	Analogový vstup kanálu B (-)
B+	Analogový vstup kanálu B (+)
GND	Uzemnění
DT	Digitální výstup sériových dat
SCK	Vstup časování sériového přenosu
VCC	Napájení

2.2.2 Popis pinů čipu HX711



Obrázek 2.6: Popis pinů čipu HX711 [11]

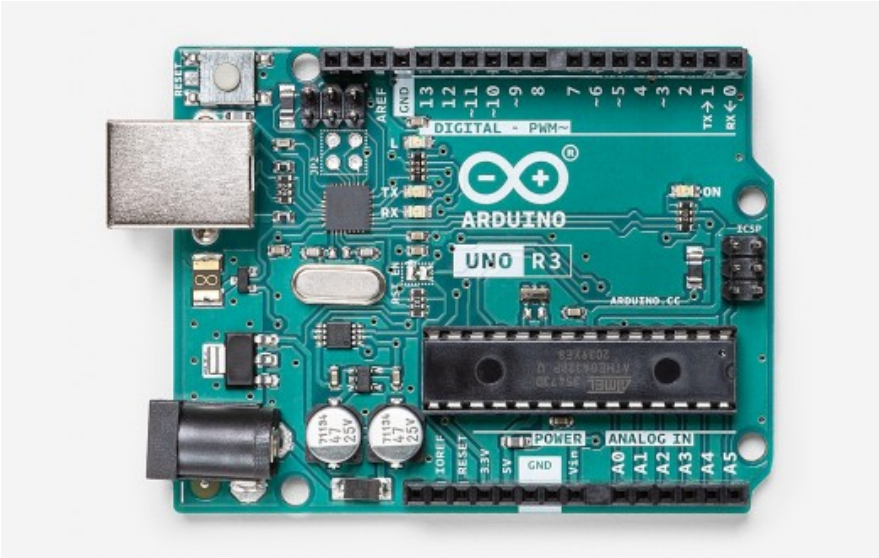
Tabulka 2.3: Popis pinů čipu HX711

Číslo pinu	Značení	Funkce	Popis
1	VSUP	Napájení	Vstup regulátoru: 2.7 ~5.5V
2	BASE	Analogový výstup	Výstup regulátoru
3	AVDD	Napájení	Analogové napájení 2.6 ~5.5V
4	VFB	Analogový vstup	Výstup regulátoru
5	AGND	Uzemnění	Analogové uzemnění
6	VBG	Analogový výstup	Výstup referenčního bypassu
7	INA-	Analogový vstup	Negativní vstup kanálu A
8	INA+	Analogový vstup	Pozitivní vstup kanálu A
9	INB-	Analogový vstup	Negativní vstup kanálu B
10	INB+	Analogový vstup	Pozitivní vstup kanálu B
11	PD_SCK	Digitální vstup	Ovládání vypnutí a vstup sériového časování
12	DOUT	Digitální výstup	Výstup sériových dat
13	XO	Digitální I/O	I/O krystalu
14	XI	Digitální vstup	I/O krytalu nebo externí vstup časování
15	RATE	Digitální vstup	Nastavení vzorkování
16	DVDD	Napájení	Digitální napájení 2.6 ~5.5V

2.3 Vývojová deska Arduino

2.3.1 Arduino UNO

Arduino UNO je malý jednodeskový počítač založený na mikročipu ATmega328P. V dnešní době je jedním z nejpoužívanějších typů programovací desky, převážně v kategorii Internet of Things. Deska je osazena čtrnácti digitálními I/O piny a šesti analogovými vstupními piny programovatelnými pomocí USB vstupu typu B a softwaru Arduino IDE. Pro napájení lze využít USB kabelu nebo externí 9V baterie.

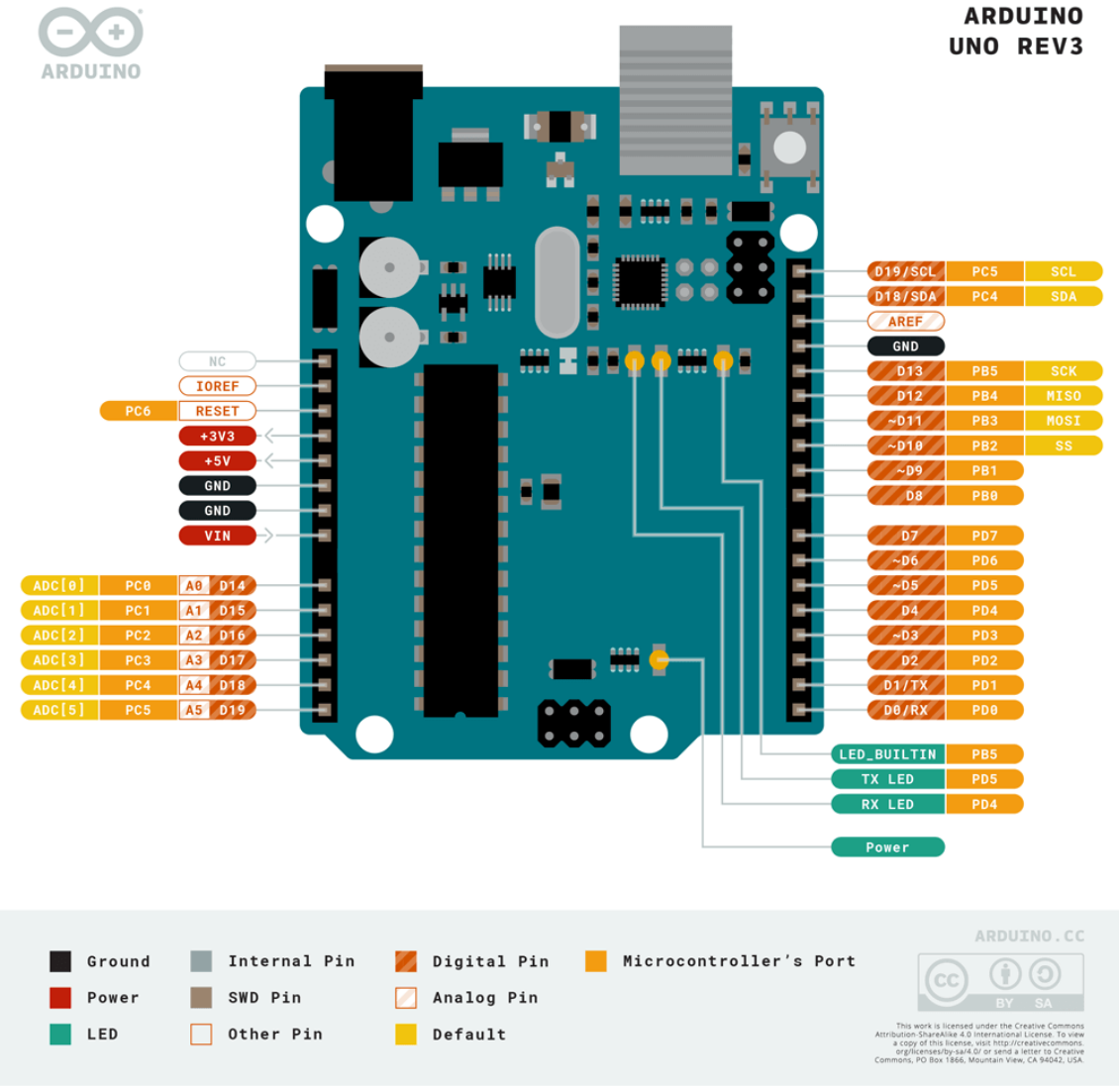


Obrázek 2.7: Vývojová deska Arduino UNO [13]

2.3.2 Jiné vývojové desky

Mimo Arduino UNO jsou vyráběny i jiné typy programovatelných desek. Pro použití na menších projektech, zejména IoT, jsou vhodné například modely Arduino Nano či Arduino Micro. Pro složitější projekty, které potřebují větší množství použitelných pinů nebo vyšší výkon je to například deska Arduino Mega.[14] Existují však i modely jiných výrobců. Příkladem můžou být mikrokontroléry jako NodeMCU, SparkFun RedBoard nebo desky Teensy.

2.3.3 Popis pinů



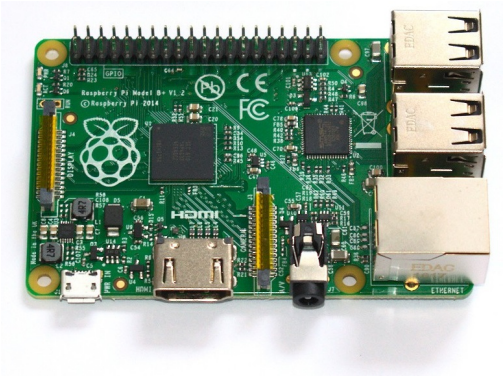
Obrázek 2.8: Popis pinů vývojové desky [13]

2.4 Minipočítač Raspberry Pi

Raspberry Pi označuje řadu jednočipových jednodeskových počítačů s deskou plošných spojů vyznačující se zejména malou velikostí a velmi širokým rozsahem možných použití. Existuje již více generací, lišících se výkonem nebo zaměřením na určité použití. Většina modelů disponuje HDMI konektorem pro připojení monitoru a USB konektory pro připojení myši, klávesnice, či jiných periferií. Počítač podporuje řadu operačních systémů, jako například různé distribuce Linuxu nebo Windows 10 IoT Core společnosti Microsoft. Primárně se však používá operační systém Raspberry Pi OS, dříve nazývaný Raspbian, který byl odvozen od linuxové distribuce Debian. Oproti vývojové desce Arduino je možné Raspberry Pi používat i jako plnohodnotný počítač, tedy jak pro ovládání, tak i pro vývoj potřebných aplikací.

2.4.1 Raspberry Pi Model B+

V rámci projektu byl využit dostupný původní model Raspberry Pi Model B+ disponující 512 MiB RAM paměti, čtyřmi USB konektory, internetovým konektorem RJ-45, konektorem HDMI a slotem pro mikro SD kartu. [15]



Obrázek 2.9: Raspberry Pi Model B+ [16]

2.4.2 Další modely Raspberry Pi

Raspberry Pi Model B a A – původní modely s pamětí 256 MB a pouze 26ti GPIO piny.

Raspberry Pi Model A+ – levnější alternativa Raspberry Pi, poslední revize původního modelu A. Od modelu B+ modely disponují 40ti GPIO piny.

Raspberry Pi 2 Model B – druhá generace Raspberry Pi s pamětí 1GB nahrazující model B+.

Raspberry Pi 3 Model B – první model třetí generace Raspberry Pi.

Raspberry Pi 3 Model A+ – kompaktnější verze třetí generace, stejného formátu jako původní model A+

Raspberry Pi 3 Model B+ – poslední revize třetí generace Raspberry Pi.

Raspberry Pi 4 Model B – čtvrtá generace Raspberry Pi s možnou pamětí 2GB, 4GB nebo 8GB. Nově disponující porty jako Micro HDMI, USB 3.0 a USB typu C.

Raspberry Pi Zero – menší a kompaktnější Raspberry Pi s pamětí 512MB. Verze Zero W navíc disponuje bezdrátovou konektivitou jako Wi-Fi a Bluetooth.

Raspberry Pi Pico – zatím nejnovější, miniaturní verze Raspberry Pi, postavená na novém čipu RP2040.

Podobně jako u desek Arduino, existuje více výrobcu podobných minipočítačů. Takovými jsou například Banana Pi, Rock Pi nebo i ASUS Tinker board.

2.5 Popis protokolů pro přenos dat

2.5.1 Komunikace mezi modulem HX711 a vývojovou deskou Arduino

Přenos dat mezi modulem a vývojovou deskou probíhá sériovou komunikací pomocí dvou vodičů, kde jeden vodič slouží pro sériové časování a druhý pro přenos dat. O komunikaci se starají piny **PD_SCK** a **DOUT** a slouží k získání dat, výběr vstupu a zisku, a ovládání vypnutí. Pokud je pin **DOUT** v aktivním stavu, signalizuje, že data nejsou připravena. Pokud je ale pin **DOUT** ve stavu 0, data jsou připravena a aplikováním určitého počtu aktivních pulzů na pin **PD_SCK** jsou data přenášena pinem **DOUT** dále, kde s každým pulzem se přenáší jeden bit dat. Po proběhnutí 24 pulzů přenášející data je následujícím pulzem pin **DOUT** opět nastaven na aktivní hodnotu 1. Celkovým počtem pulzů na pin **PD_SCK** a **DOUT** je možné nastavit zisk a používaný kanál. [11]

Tabulka 2.4: Výběr zisku a kanálu čipu HX711

Počet PD_SCK pulzů	Vstupní kanál	Zisk
25	A	128
26	B	32
27	A	64

2.5.2 Komunikace mezi vývojovou deskou Arduino a počítačem

Vývojová deska Arduino disponuje řadou komunikačních periferií používající různé komunikační protokoly. Příkladem je sériový protokol **I2C**, používaný především pro moduly a senzory dovolující připojení až 128 zařízení. Dále také protokol **SPI** užívaný spíše pro komunikaci mezi mikrokontroléry kvůli jeho rychlosti.[17]

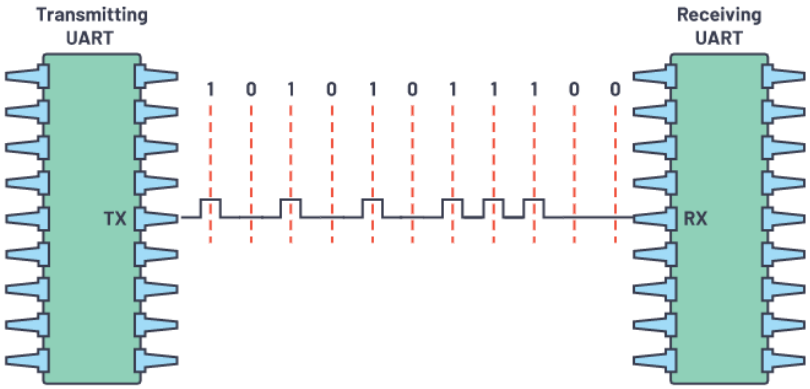
Pro komunikaci s počítačem je však použit protokol **UART**. Deska obsahuje USB-to-Serial převodník, umožňující jednoduché připojení desky k počítači pomocí USB portu. S připojeným Arduinem je poté možné komunikovat, jako by byl připojen na sériovém portu.[18]

Protokol **UART** dovoluje oboustrannou komunikaci pomocí dvou signálů:

Tx (Transmitted Serial Data) – Přenášená sériová data

Rx (Recieved Serial Data) – Přijatá sériová data

Rozhraní UART nepoužívá k synchronizaci časovací signál (je asynchronní), ale vygeneruje bitový proud podle nastavené přenosové rychlosti (**baud rate**). Ta musí být nastavena na stejnou hodnotu jak na vysílači, tak na přijímači.



Obrázek 2.10: Sériový přenos dat [18]

Původní data v paralelní formě jsou převedena na sériová a pro přenos uspořádána do paketů. Přenos dat probíha ve formě UART paketů skládajících se ze čtyř částí:

Start bit – jeden bit, signalizující začátek přenosu paketu. Po přečtení paketu začne přijímač číst bity v nastavené přenosové rychlosti (baud rate).

Datový rámec – obsahuje přenášená data. Může být v rozsahu 5 až 8 bitů. Pokud není použita parita, může být až 9 bitů.

Parita – bit pro kontrolu úspěšnosti přenosu. Protokol po přenosu spočítá počet přenesených bitů s logickou hodnotou 1. Pokud je bit parity s logickou hodnotou 0, tedy sudý, měl by být počet bitů s hodnotou 1 také sudý. Naopak pokud je bit parity s hodnotou 1, měl by být počet bitů s hodnotou 1 v datovém rámci lichý.

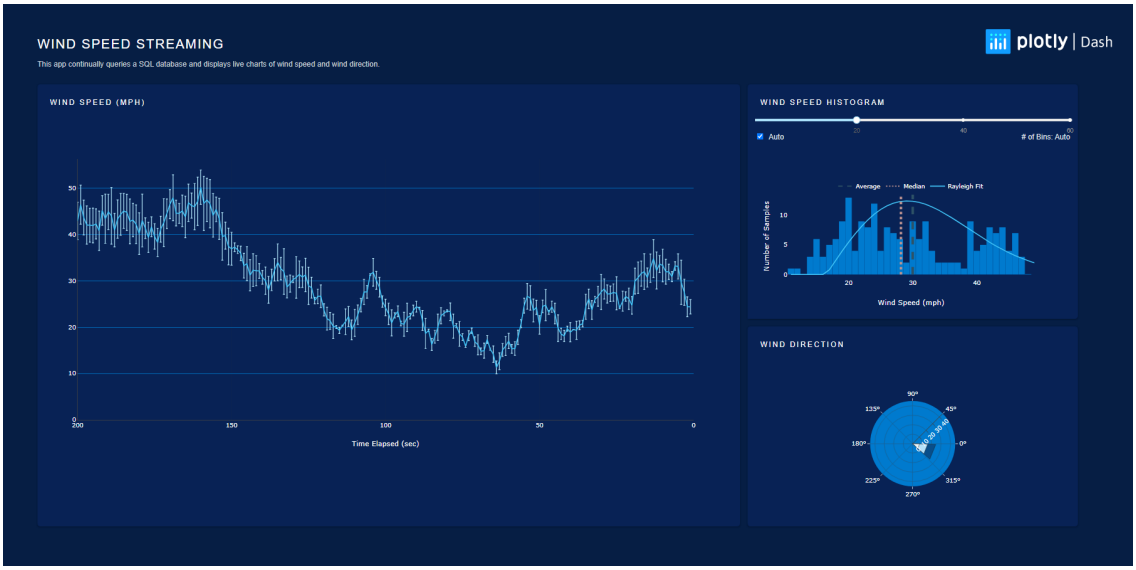
Stop bity – signalizující konec přenosu. Může být jeden nebo dva.

Obrázek 2.11: UART Paket [18]

2.6 Použitý software a služby

2.6.1 Plotly Dash

Firma Plotly se zabývá vývojem online nástrojů pro vizualizaci a analytiku dat. Vyvíjí aplikace pro online tvoření grafů a různých statistik, ale také knihovny pro tvorbu vlastních aplikací s využitím různých programovacích jazyků jako Python nebo MATLAB. Dash označuje open-source Python aplikační rámec (framework) pro vytváření analytických aplikací s využitím na webu. Dovoluje širokou customizaci a úpravu uživatelského prostředí a vizualizace dat. K aplikacím se přistupuje skrze webový prohlížeč, to také značí jednoduchý přístup i z mobilních zařízení a dalších platforem. Dash je postaven na mikro webovém frameworku Flask a JavaScriptových knihovnách Plotly.js a React.js. [19]



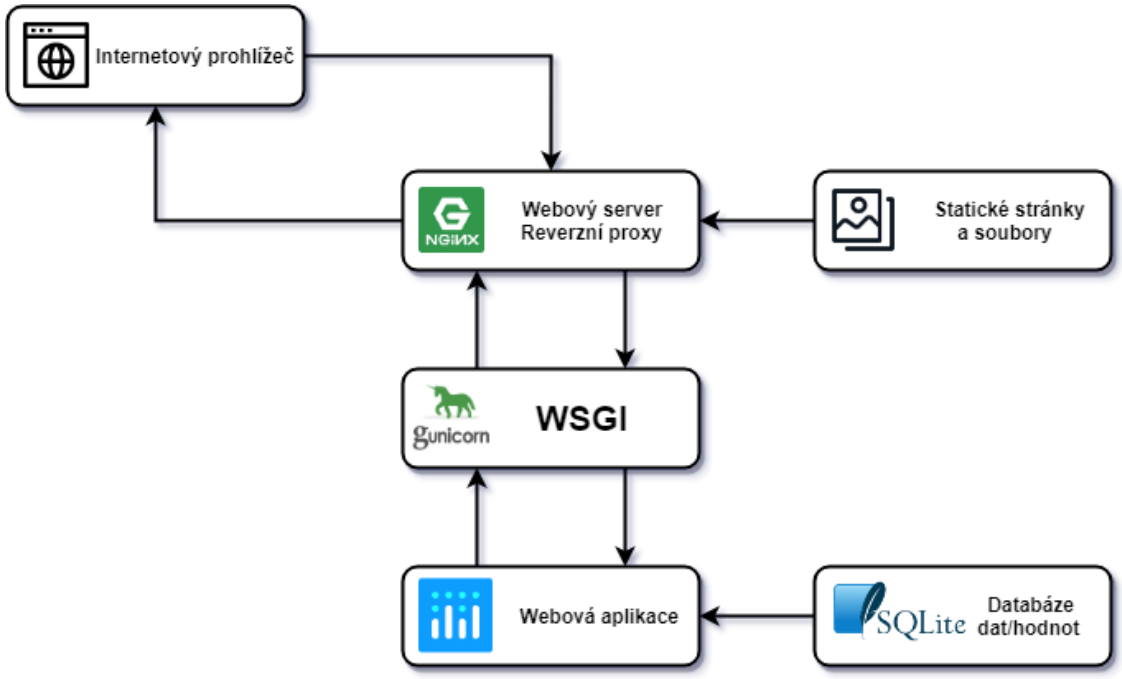
Obrázek 2.12: Příklad Dash aplikace [20]

2.6.2 Gunicorn

Gunicorn neboli „Green Unicorn“, je název pro takzvaný WSGI HTTP server pro UNIXové prostředí napsaný v jazyce Python. WSGI neboli Web Server Gateway Interface je rozhraní mezi webovým serverem a webovou aplikací, definující jak mezi sebou webový server a webová aplikace komunikují. Green Unicorn byl odvozen od projektu Unicorn, který je používán pro webové aplikace psané v programovacím jazyce Ruby. [21]

2.6.3 Nginx

Nginx je open-source webový server vyznačující se svou stabilitou a vysokým výkonem. Zároveň pracuje i jako reverzní proxy, tedy proxy server v lokální síti, který převádí dotazy z internetu dále na patřičný server, například s požadovanou webovou aplikací či službou. Bez použití WSGI serveru může být konfigurován jako webový server pro statické webové stránky a soubory. [23]



Obrázek 2.13: Dotaz na webovou aplikaci

Obrázek popisuje průběh dotazu z internetu na webovou aplikaci. Dotaz klienta dorazí na webový server Nginx kde jej reverzní proxy přesměruje na WSGI server Gunicorn. Ten zavolá webovou aplikaci a odpověď vrací zpět. Webový server Nginx poté odpovídá zpět klientovy.

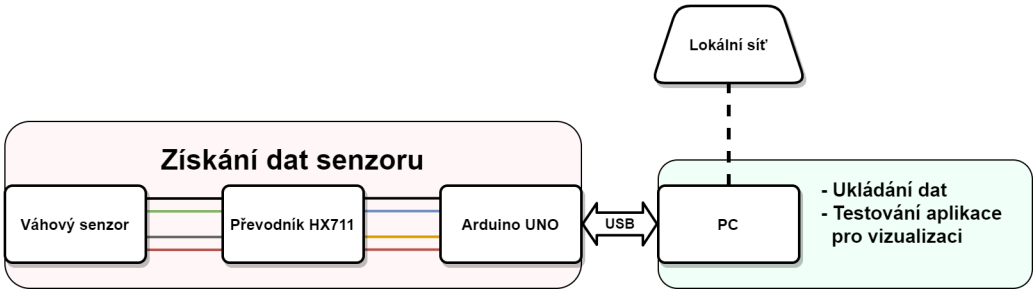
Kapitola 3

Sběr a ukládání dat senzoru

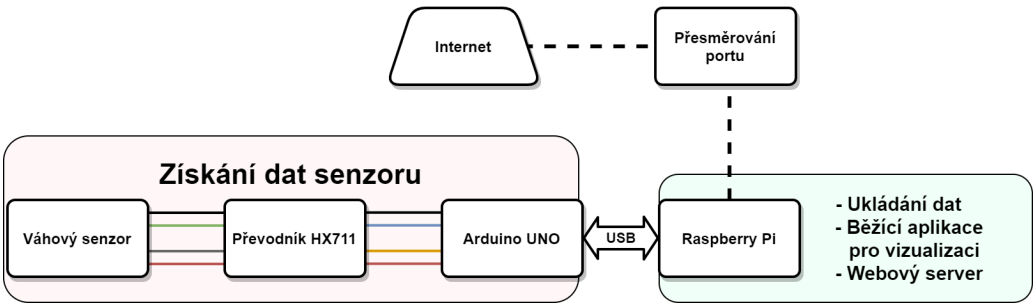
3.1 Schéma zapojení

Při práci bylo využito dvou zapojení. Vývojová deska s převodníkem a senzorem byla nejprve připojena k počítači s systémem Windows 10 kde byly vytvořeny potřebné kódy a byla testována funkčnost na lokální síti.

Senzor byl poté zapojen do zařízení Raspberry Pi kde byla aplikace spuštěna a nastaven webový server pro přístup k internetu.



Obrázek 3.1: Zapojení - Lokální síť

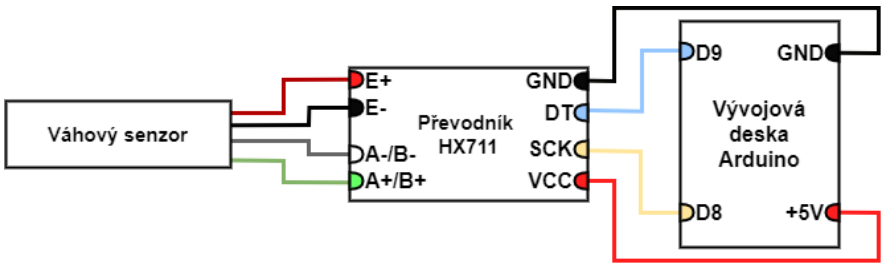


Obrázek 3.2: Zapojení - Internet

3.2 Získání hmotnostních dat senzoru

3.2.1 Zapojení senzoru a převodníku k vývojové desce

Prvním krokem je propojení součástek, tedy připojení váhového senzoru k převodníku HX711 a toho následně k programovatelné desce Arduino. Podle dokumentace je použito k napájení červeně a černě značeného vodiče senzoru připojených na excitační výstupy značených jako E+ a E- na AD převodníku. Pro přenos signálu je využito vodičů značených bílou a zelenou barvou připojených na vybraný kanál převodníku (A+ A- nebo B+ B-). Dále jsou piny značené jako VCC a GND na HX711 spojeny s napájením a uzemněním na desce Arduino. Piny značené jako SCK a DT jsou připojeny na zvolené digitální piny D8 a D9 desky Arduino.



Obrázek 3.3: Zapojení senzoru

3.2.2 Kód pro čtení dat senzoru

Programování vývojové desky probíhá pomocí softwaru Arduino IDE. Arduino IDE je aplikace napsaná ve funkcích jazyků C a C++, používá se k vývoji programů pro programovatelné desky a je možné ji stáhnout zdarma z oficiálních stránek Arduino. Pro práci s HX711 je nutné stáhnout příslušnou knihovnu. Navigací přes „Tools → Manage Libraries“ nebo klávesovou zkratku CTRL+Shift+I se otevře správa knihoven a vyhledáním „HX711“ se zobrazí knihovny pro tento modul. Knihovny je také možné instalovat ručně stáhnutím souborů knihovny a jejich vložením do adresáře „Dokumenty → Arduino → libraries“. Takto byla nainstalována použitá knihovna[25].

Mimo import knihoven a definici používaných proměnných a objektů se kód dělí do dvou částí, těmi jsou **setup**, kde probíhá první nastavení při zapnutí programu, a **loop**, kde se nachází kód, který je neustále opakován.

V kódu byl podle knihovny vytvořen objekt pro váhu **scale(DOUT, CLK)**, kde parametry DOUT a CLK jsou proměnné s číslem, odpovídajícím pinu kde je připojen daný vodič z modulu HX711. DOUT je nastaven na pin pro data a CLK je nastaven na pin pro sériové časování. Dále byly vytvořeny proměnné pro kalibrační faktor a pro výpočet rozdílu.

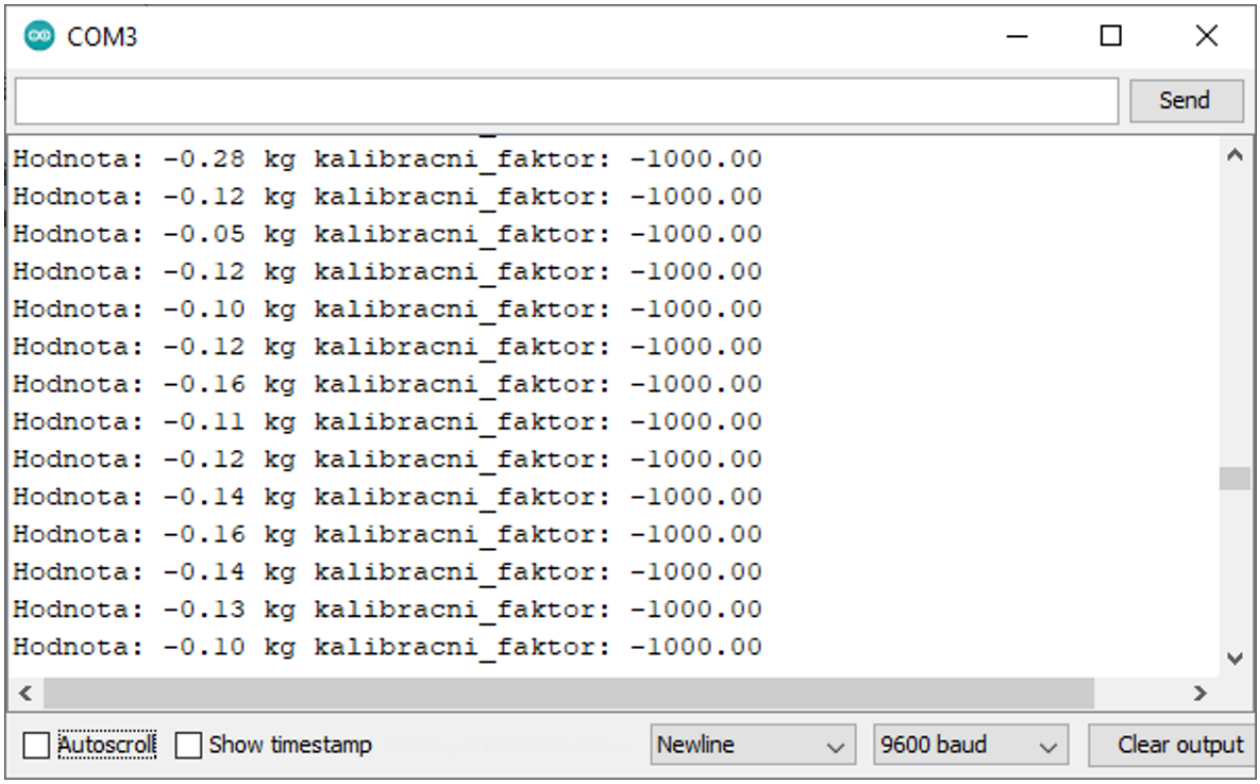
Ve funkci **setup** je definováno sériové vysílání funkcí **Serial.begin** a čtení je nastaveno na hodnotu 0 spuštěním funkcí **scale.set_scale()** a **scale.tare()**

Ve funkci **loop** je nastaven aplikován kalibrační faktor vložením jeho hodnoty jako parametr funkci **scale.set_scale(calibration_factor)**. Tím je kalibrována správná funkčnost a citlivost senzoru. V podmínce **if(Serial.available())**, která kontroluje dostupnost sériového vstupu, je nastavena možná změna kalibračního faktoru za běhu programu zadáváním příslušných znaků. V loop funkci rovněž probíhal výpočet váhového rozdílu a data byly následně vypsána funkcí **Serial.println** do sériového monitoru.

Aby bylo možné kód nahrát do vývojové desky, je navigací „Tools → Board“ nastavena deska Arduino UNO a navigací „Tools → PORT“ je nastaven port, ke kterému je deska připojena. Stiskem tlačítka „Upload“ je kód nahrán do desky.

3.2.3 Testování senzoru

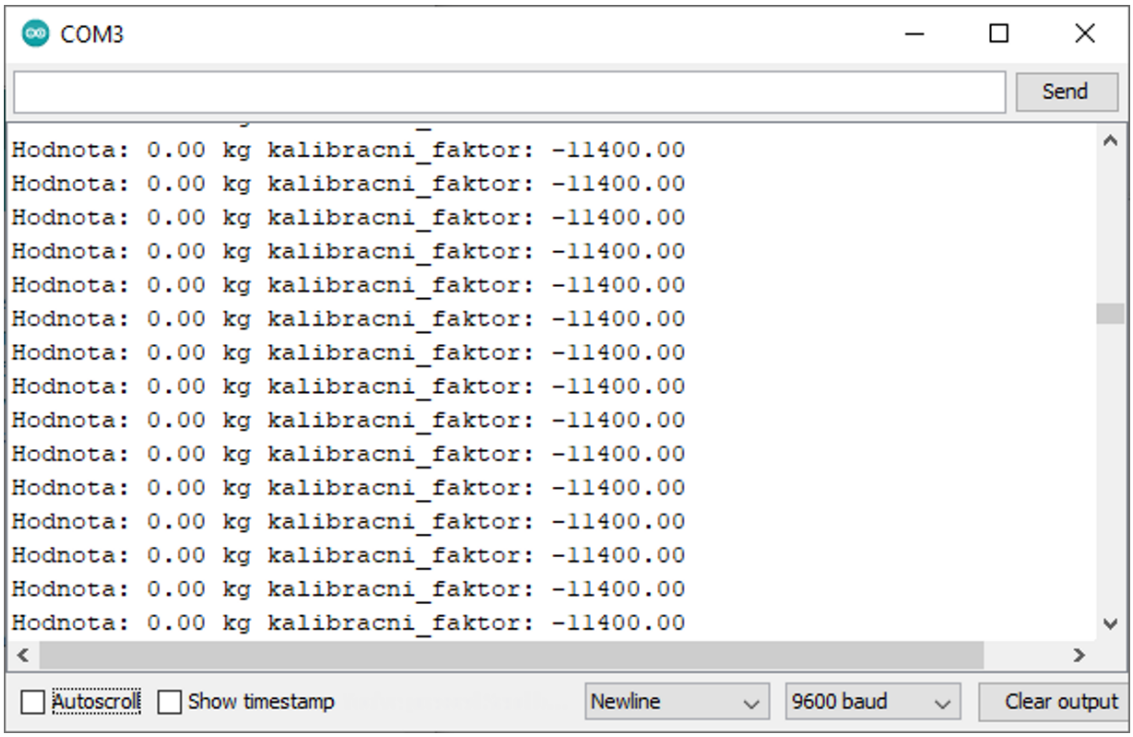
Při prvním měření vypisované hodnoty neodpovídaly hodnotám očekávaným a nereagovali na působenou sílu na senzor. Na obrázku lze vidět, že i v klidném stavu se hodnoty náhodně liší v rozsahu až 23 nastavených jednotek.



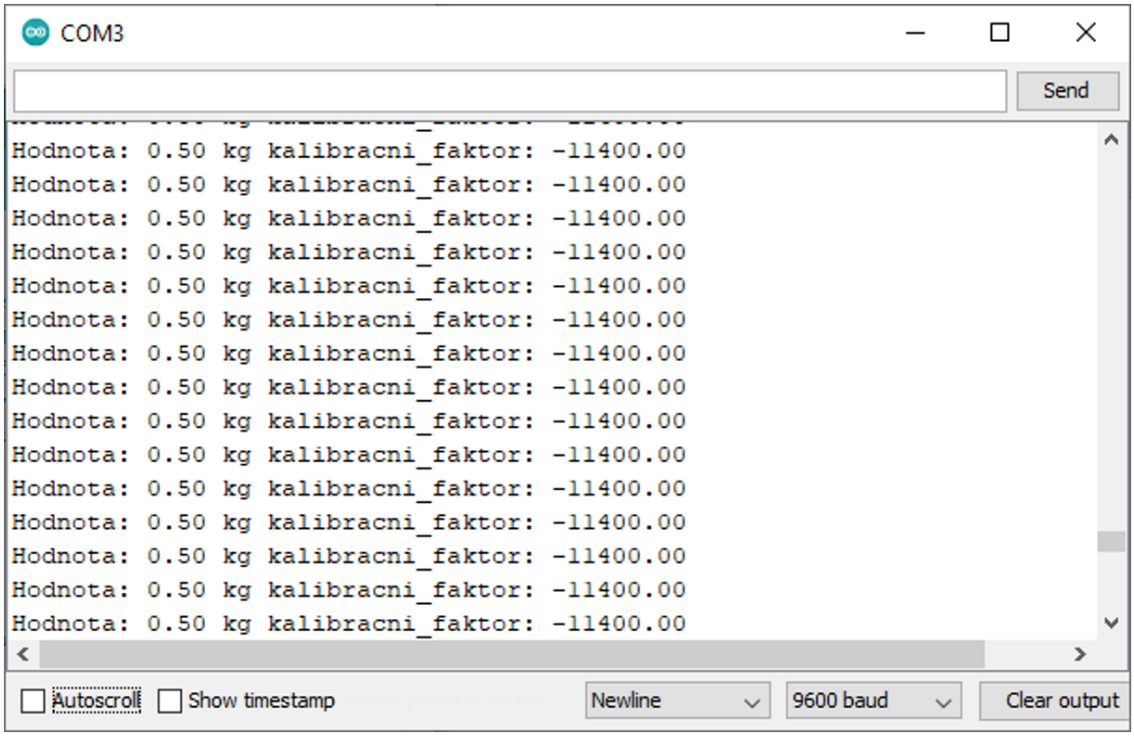
Obrázek 3.4: Nesprávné hodnoty

Při ověření funkce senzor se opět vychází z principu Wheatstonova můstku, tedy všechny odpory senzoru by měly být vyrovnané. Na připojeném a napájeném senzoru jsou postupně změřeny odpory na dvojicích pinů A+ E+, A- E-, A+ E-, A- E+. Všechny naměřené hodnoty byly téměř identické. Výsledek každého měření se velmi blížil hodnotě 290Ω.

Dále bylo na výstupech signálu (A+ A-) naměřeno velmi nízké napětí v milivoltech a při aplikaci zátěže na váhový senzor bylo možné zaznamenat malý nárůst napětí. Použitý hmotnostní senzor tedy problémem nebyl. Otestováno bylo i několik různých HX711 knihoven a použití různé digitálních a analogových pinů na desce Arduino. Závěrem tedy bylo, že se jedná o vadný zesilovač HX711. Po prostudování dokumentace použité knihovny byl nalezen kód pro přepnutí používaného kanálu na kanál B pomocí příkazu **scale.set_gain(32)**, který nastaví zisk či zesílení převodníku na hodnotu 32, kterou umí pouze kanál B. Čtení hodnot následně začalo fungovat správně. Chybou byl tedy nefunkční kanál A na převodníku HX711.



Obrázek 3.5: Kalibrované měření bez zátěže



Obrázek 3.6: Kalibrované měření s zátěží 500g

3.3 Aplikace pro ukládání dat

Pro snadnější odečítání hodnot hmotnosti bylo nutné pozměnit kód běžící na desce Arduino. Místo textového doprovodu a popisu jednotky byl výpis dat změněn na naměřenou hodnotu a vypočítanou hodnotu změny oproti hodnotě předchozí. Tyto dvě hodnoty byly vypisovány odděleny čárkou a s každým novým měřením na nový řádek.

3.3.1 RealTerm

Jako software pro odečítání a ukládání byl zpočátku zvolen program RealTerm, který umožňuje postupné zapisování řádků sériového monitoru do CSV souboru, tedy souboru hodnot oddělených čárkou. Program požaduje nastavení připojeného sériového portu a souboru pro zapisování. Bylo možné sledovat odečítání správných hodnot, soubor však stále zůstával prázdný. Vyzkoušeno bylo různých nastavení, využití tohoto programu však ve výsledku nebylo ideální.

3.3.2 Python aplikace

Jako lepší řešení bylo zvoleno využít programovacího jazyku Python a knihovny **pyserial**. Knihovna pyserial je rozšíření jazyku Python pro práci se sériovými porty. Program byl nastaven na port používaný připojeným arduinem a pomocí funkce readline bylo možné po řádcích korektně odečítat data vypisována na sériovém portu. Funkcí **writer** knihovny csv byla tyto data následně zapisována do CSV souboru.

Ukládání do CSV souboru se však projevilo jako neideální pro následné čtení a práci s daty. Proto bylo zvoleno přejít na databázový systém SQLite. Jazyk Python podporuje práci s SQLite pomocí knihovny **sqlite3** a tím vytvářet a pracovat s databázemi pomocí jazyku SQL. Program na desce Arduino byl dále pozměněn tak, aby vypisoval do sériového monitoru pouze naměřenou hodnotu na každý nový řádek bez dopočítaného rozdílu. Ten je možné dopočítat v Python programu pro zapisování zároveň s jinými daty. Do databáze je tedy zapisována naměřená hodnota, čas a datum kdy byla změřena, a vypočítaný rozdíl oproti předchozí hodnotě. Vytvoření tabulky a zápis do databáze je řešen SQL příkazy spouštěnými funkcí **execute**.

3.3.3 Popis funkcí zapisovací aplikace

Výsledný zapisovací program obsahuje dvě vlastní definované funkce. Funkce s názvem **read _serial** se stará o čtení řádků sériového monitoru. Řádek načte pomocí funkce readline a dekóduje podle kódování ASCII funkcí **decode**. Dekódovaný řádek je uložen do proměnné **line** jako řetězec. Později bylo zjištěno, že načtený řádek obsahuje zbytečné znaky. To bylo vyřešeno přidáním funkce **rstrip**, která odstraní přebytečné znaky na konci načteného řetězce.

```
def read_serial():
    line = ser.readline().decode('ascii').rstrip()
    print("read")
    return line
```

Listing 3.1: Definovaná čtecí funkce

Zápis do tabulky databáze je řešen definovanou funkcí **db_write**, která přijímá čtyři parametry, a to jednotlivá data určená k zápisu do každého sloupce. Funkce se připojí do názvem určeného databázového souboru a vytvoří kurzor pro její procházení. Zavoláním funkce **execute** poté provede definovaný SQL příkaz.

```
def db_write(time, w_data, d_data, date):
    connection = sqlite3.connect('weightData.db')
    c = connection.cursor()

    c.execute(""" CREATE TABLE IF NOT EXISTS weightdata (time TEXT, weight TEXT,
        difference TEXT, date TEXT)""")
    c.execute(""" INSERT INTO weightdata (time, weight, difference, date)
    VALUES (?, ?, ?, ?)""", (time, w_data, d_data, date))

    connection.commit()
    c.close()
    connection.close()
```

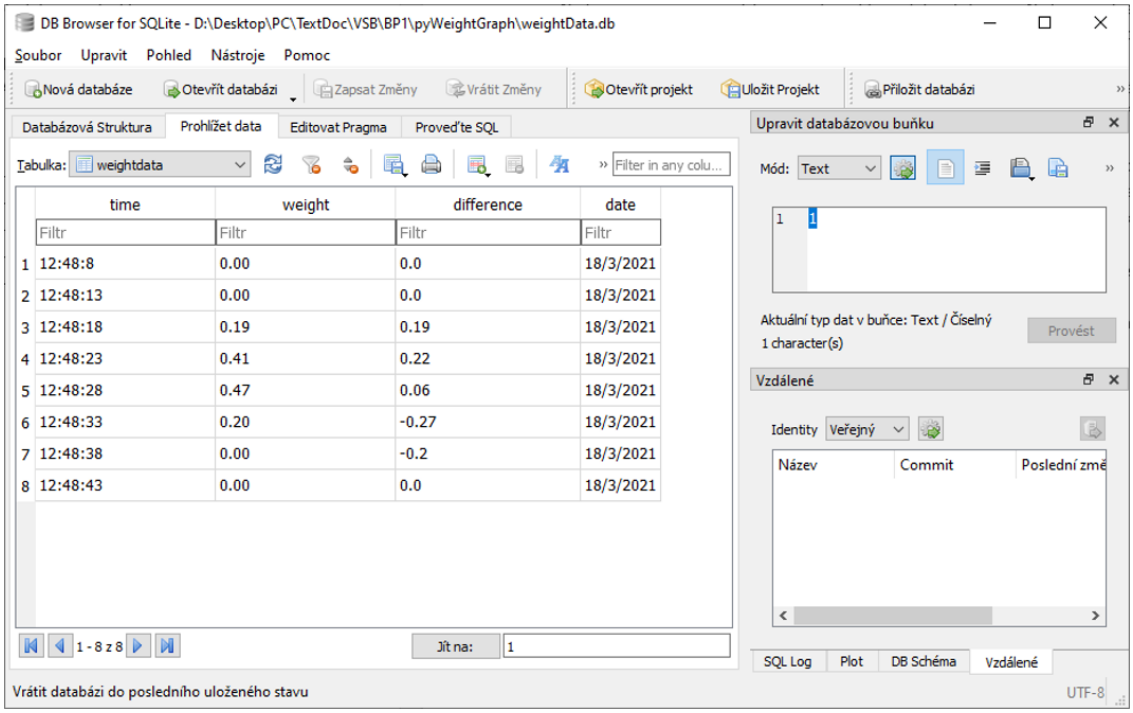
Listing 3.2: Definovaná zapisovací funkce

První SQL příkaz, pokud již neexistuje, vytvoří novou databázovou tabulku s názvem weightdata v nastaveném souboru se sloupci pro čas měření, naměřenou váhu, rozdíl oproti předchozí hodnotě a datum měření. Následující SQL příkaz do tabulky vloží v programu připravená data.

Poslední částí zapisovacího programu je cyklus **while True**, který opakovaně využívá definovaných funkcí. Program v cyklu načte data a zkontroluje, zda nejsou prázdná. Pokud načtení proběhne v pořádku, je z načtené hodnoty vypočítán rozdíl měření a zjištěn čas a datum, kdy měření proběhlo. Data jsou poté vložena do tabulky definovanou zapisovací funkcí.

Jelikož funkce běží opakovaně, je nutné limitovat rychlost opakování měření funkcí **sleep**, ideálně na hodnotu spánku nastavenou pro samotné vypisování hodnot na desce Arduino.

Vytvoření databázového souboru bylo provedeno pomocí volně dostupného softwaru SQLite Database Browser. Zde je taktéž možné zobrazit tabulky databáze vytvořené zapisovacím programem a jejich zapsaná data. Obrázek ukazuje otevřený databázový soubor **weightData.db** obsahující tabulku **weightdata** se čtyřmi sloupci.



Obrázek 3.7: Databázová tabulka

3.3.4 Zapisování na minipočítači Raspberry Pi

Do počítače byl flash diskem přenesen připravený program pro čtení dat a databázový soubor. Pro jejich spuštění bylo nutné nainstalovat potřebné knihovny. K instalaci byl použit správce modulů Pip. Instalace některých knihoven však trvala velmi dlouhou dobu, někdy i několik hodin. Důvodem mohlo být použití zastaralého modelu Raspberry Pi, nebo příliš náročného systému. Proto byla zvážena přeinstalace na méně náročnou verzi operačního systému s názvem Raspberry Pi OS Lite. To však nakonec nebylo nutné.

Jelikož dříve připravené programy používají programovací jazyk Python verze 3, místo příkazu pro instalaci pip bylo potřeba použít příkaz pip3. Instalace knihoven po použití příkazu pip3 byla značně rychlejší, trvala v rámci pouze několika minut.

Po instalaci potřebných knihoven byl pomocí editoru nano v načítacím programu nastaven používaný sériový port. Ten byl zjištěn spuštěním příkazu **dmesg | grep tty** v terminálu Raspberry Pi. Program pro čtení hodnot byl nyní připraven ke spuštění. Čtení a zápis dat senzoru do databáze může být zahájen příkazem python se jménem souboru čtecího kódu.

Kapitola 4

Vizualizace naměřených dat

4.1 Aplikace pro tvorbu grafů

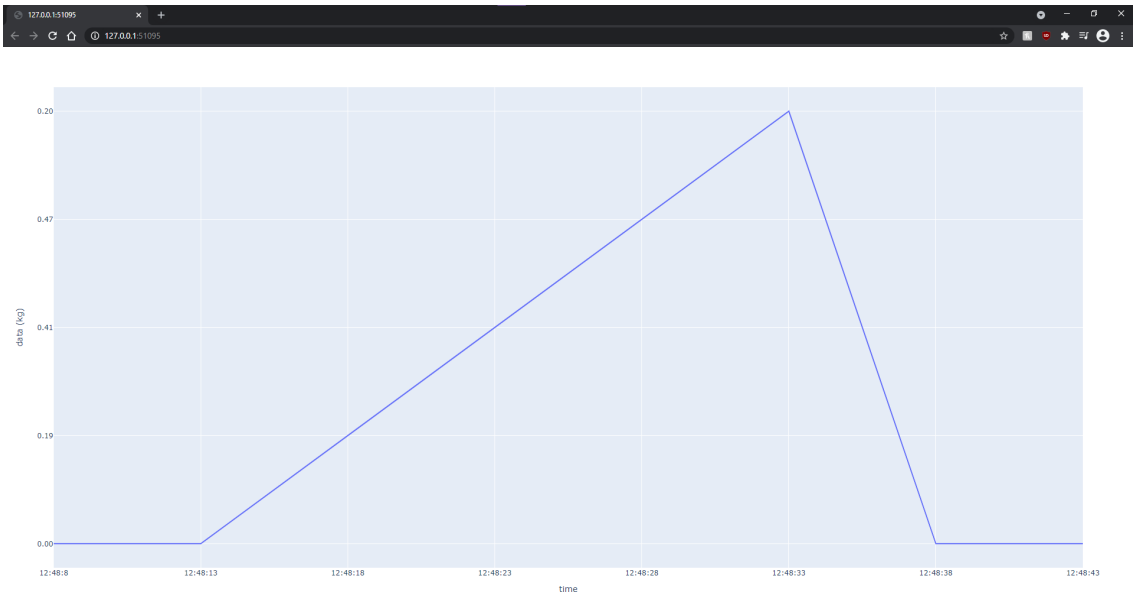
4.1.1 Základní načtení a vykreslení dat

Pro aplikaci s grafy byl rovněž zvolen programovací jazyk Python a jeho rozšíření knihovnou Plotly. Ta umožňuje tvorbu kódu pro vykreslování dat do grafů podle specifikace uživatele. Nabízí široký výběr různých typů vykreslení dat do obyčejných grafů jako sloupcových a spojnicových, ale i 2D a 3D vizualizací, tabulek, teplotních map, animací a mnoho dalších. Výstupem programu využívající knihovnu Plotly je graf či vizualizace v jazyku HTML pro otevření v internetovém prohlížeči. Aplikace nejprve musí načíst potřebná uložená data, to bylo zpočátku řešeno knihovnou csv a funkcí **read_csv** se symbolem čárky jako děličem. Po přechodu na databázový systém SQLite byly data po nastavení databázového souboru načítány spuštěním SQL příkazu funkcí **execute**, která pro kurzor označí celou tabulku a dále pomocí funkce **fetchall** uloží celý obsah tabulky do proměnné **dbdata**.

```
c.execute(""" SELECT * from weightdata """)
dbdata = c.fetchall()
```

Listing 4.1: Kód pro získání dat tabulky

Načtená data jsou uspořádána do sloupců podle zdrojové tabulky a uložena v proměnné datového typu list. Z této proměnné se rozdělením listu každý sloupec uložil do vlastní proměnné. Tímto byly získány čtyři různé proměnné pro naměřené hodnoty, datum měření, čas měření a rozdíl oproti měření předchozímu. Pomocí knihovny **Plotly** je poté možné vytvořit sloupcový graf načtených hodnot přiřazením proměnné vybraných dat na osy x a y grafu a jeho zobrazení funkcí **show**. Po spuštění programu se otevře nastavený výchozí webový prohlížeč na IP adrese *localhost* s vykresleným grafem.



Obrázek 4.1: Vykreslený graf v prohlížeči

4.1.2 Úprava osy grafu

První viditelnou chybou výstupu kódu jsou data poskládaná na vertikální ose Y pro hodnotu váhy v kilogramech. Po bližší inspekci a porovnáním s pořadím dat uložených v databázové tabulce si lze všimnout, že s každou nově objevující se hodnotou dojde k přidání hodnoty na nejvyšší bod vertikální osy. Jako řešení bylo zváženo uspořádaní proměnných s hodnotami, to by se ale projevilo jako obtížné, protože by došlo k přeházení časových hodnot. Problém byl vyřešen použitím funkce **update_yaxes**, knihovna Plotly totiž dovoluje nastavení různých typů os grafu. Osa Y byla automaticky nastavena na kategorické uspořádání, tedy s každou novou hodnotou vytvořila na ose novou kategorii. Zavoláním funkce **update_yaxes** s parametrem pro lineární typ osy se vertikální osa uspořádala podle velikosti hodnoty.

```
fig = px.line(x=time_data, y=weight_data, labels={'x':'time', 'y':'data (kg)'})
fig.update_yaxes(type = "linear")
fig.show()
```

Listing 4.2: Kód pro úpravu Y osy grafu

Kód popisuje vytvoření nového objektu spojnicového grafu s přiřazením časových dat pro osu X a hmotnostních dat pro osu Y. Osám jsou přiřazeny také popisky podle reprezentovaných dat. Vertikální osa je dále nastavena na lineární typ funkcí **update_yaxes** a graf je vykreslen funkcí **show**.

Obrázek 4.2: Vykreslený graf s upravenou osou Y

Takto vygenerovaný graf je již zároveň velmi interaktivní, najetím kurzorem myši na vybraný bod grafu zobrazí jeho přesnější informace. Kliknutím a táhnutím myší lze označit plocha, na kterou se graf zaměří. V pravém horním rohu stránky se nachází ovládací panel, dovolující stáhnutí aktuálního grafu jako obrázek, dále libovolné přiblížení či oddálení grafu, nebo zapnutí pomocných čar.

4.1.3 Dynamické vykreslování dat

Dosud vygenerovaný HTML kód graf vykreslí jednorázově a zobrazí jej na statické webové stránce. Program je tedy nutné upravit tak, aby byl schopen aktualizovat načtená data a podle toho upravit vykreslený graf. Pro tento úkol je použita knihovna **Dash**, která slouží k vytvoření aplikace, využívající grafů tvořených knihovnou **Plotly**. Aby aplikace byla schopna načítat data z databáze, byla definována funkce **get_db_data** s parametrem jména databázového souboru. Funkce se stará o načtení databázové tabulky dříve popisovaným SQL příkazem a její roztřídění do datových proměnných. Samotná Dash aplikace se skládá ze dvou částí. První je část s názvem **layout**, která určuje, jak výsledná aplikace a její uživatelské prostředí vypadá. Druhá část s názvem **callback** popisuje její funkce a interaktivitu. Pro úpravu části **layout** je potřeba mimo knihovnu Dash importovat další její potřebné komponenty. Byla importována knihovna **dash_core_components**, ta dovoluje tvorbu vlastního uživatelského prostředí, a hlavně vkládat do aplikace grafy knihovny Plotly. Dále byla importována knihovna **dash_html_components** umožňující práci s HTML prvky.

4.1.4 Layout

Layout aplikace byl definován HTML prvkem Div obsahujícím nadpis, graf a definovaný interval. ***Interval*** je komponent, který periodicky volá definovaný **callback** programu a tím řídí, jak často se bude aplikace aktualizovat.

```
app.layout = html.Div(
    [
        html.H4('Weight Sensor Graph'),
        dcc.Graph(id='live-update-graph'),
        dcc.Interval(
            id='interval-component',
            interval=1000,
            n_intervals=0
        ),
    ]
)
```

Listing 4.3: Definice layoutu aplikace

4.1.5 Callback

Callback aplikace obsahuje vlastní definovanou funkci **update_graph_live**. Ta nejprve zavolá načítací funkci **get_db_data** a uloží si její výpis do datových proměnných. Dále následuje definice grafů aplikace. Jelikož se pracuje s více druhů dat, je vhodné připravit aplikaci pro více grafů. To je možné realizovat například vytvořením více objektů grafů. V tomto případě bylo ale zvoleno použít funkce **make_subplots** knihovny Plotly pro tvorbu skupin grafů v jednom objektu. Jejími zadanými parametry jsou počet řádků a počet sloupců. Tímto je tedy možné vytvářet libovolnou tabulku různých definovaných grafů a vizualizací. Jednotlivé grafy se poté do objektu vkládají funkcí **append_trace** s parametry grafu a souřadnicemi řádku a sloupce jeho umístění ve vytvořené tabulce grafů.

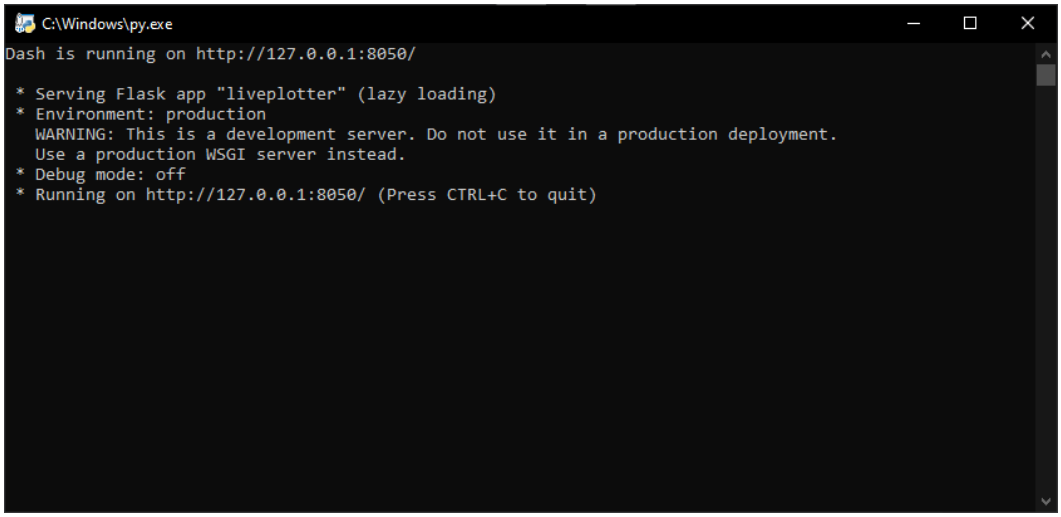
4.1.6 Spouštěcí funkce

Poslední funkce programu o spuštění aplikace. Funkce **run_server** spustí aplikaci na adrese *localhost* s výchozím portem 8050.

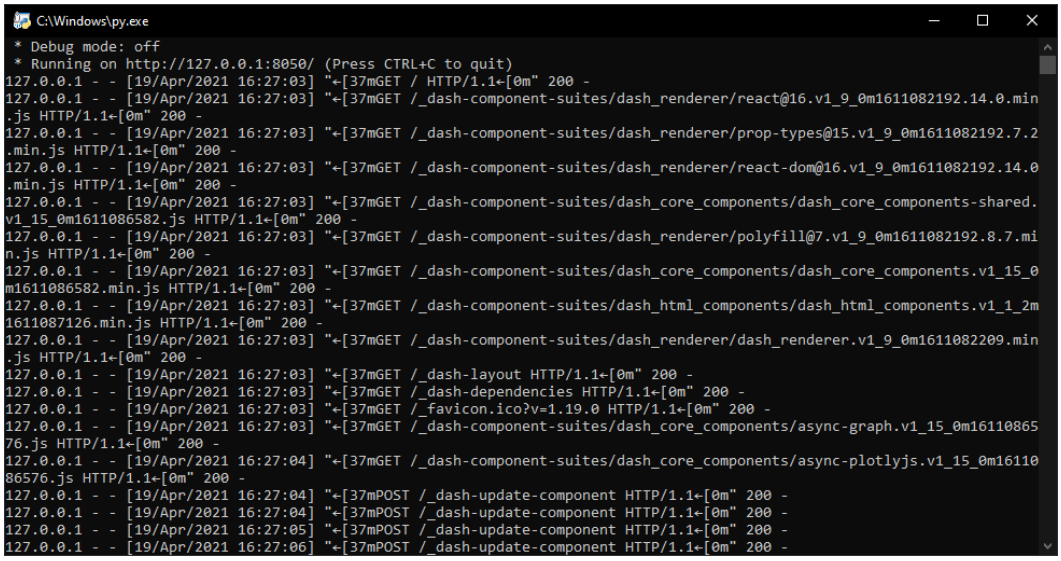
```
if __name__ == '__main__':
    app.run_server()
```

Listing 4.4: Spouštěcí funkce aplikace

Po spuštění Pythonového souboru programu se objeví konzole popisující aktivitu aplikace. Dash aplikace je spuštěna na *localhost* adrese s portem 8050 a čeká na připojení. Pokud dojde k načtení adresy v internetovém prohlížeči, konzole začne vypisovat načítající se komponenty a s každou nastavenou aktualizací jsou vypsány informace o komponentu **dash-update**. Konzole však vypisuje i varování, informující o tom, že se jedná pouze o vývojový server a pro další používání aplikace, bude potřeba nastavení WSGI serveru.



Obrázek 4.3: Konzole běžící aplikace



Obrázek 4.4: Konzole po otevření aplikace v prohlížeči

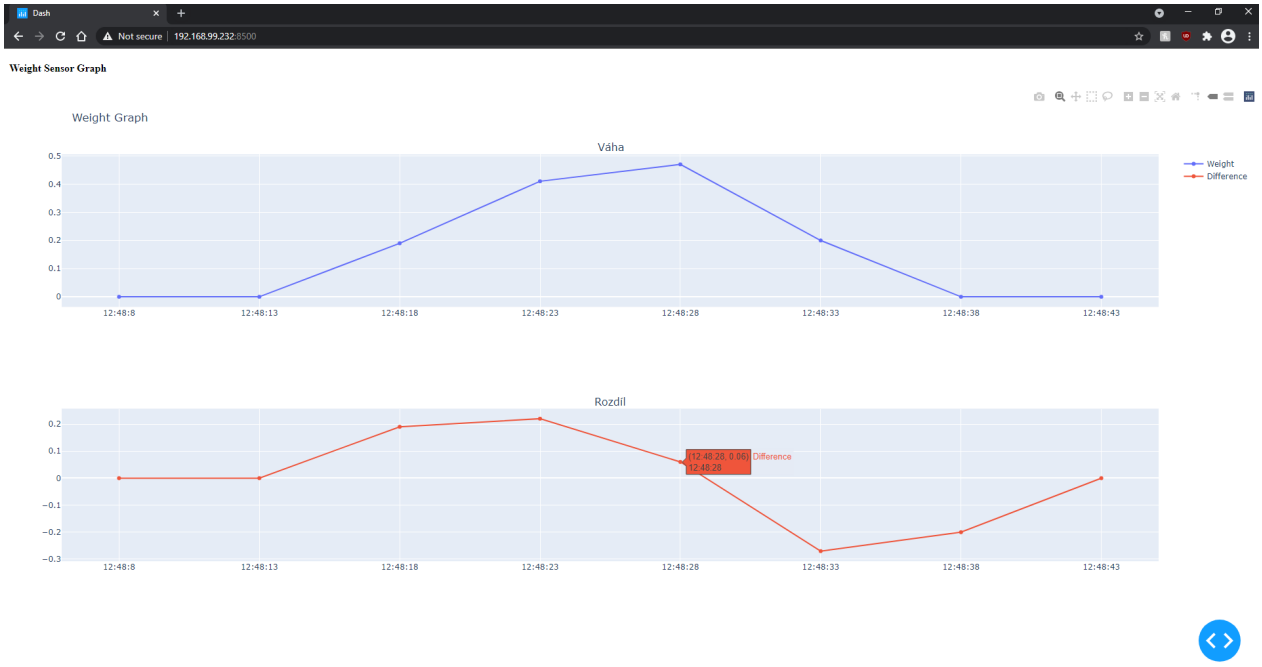
Takto spuštěná aplikace je nyní schopna v reálném čase aktualizovat graf podle periodicky načítaných dat z databáze. Pokud se tedy spuštěn i dříve popsaný program pro zápis dat, bude možné pozorovat změny v grafu a sledovat nově přidaná data.

4.1.7 Testování aplikace na lokální síti

Běžící aplikace je schopna dynamického vykreslení dat do grafu. Dostupná je však pouze na lokální adrese zařízení, na kterém byl program spuštěn. Kód programu je však možné upravit tak, aby byla dostupná i pro ostatní zařízení připojená do lokální sítě. Spouštěcí funkce **run_server** disponuje řadou nastavitelných parametrů. Nastavením boolean hodnoty parametru **debug** je možné aktivovat vývojářské nástroje. Adresa běžícího serveru je však nastavena přiřazením IP adresy zařízení v lokální síti do parametru **host**. Dodatečně je také možné změnit port, na kterém aplikace běží zadáním zvoleného čísla do parametru **port**.

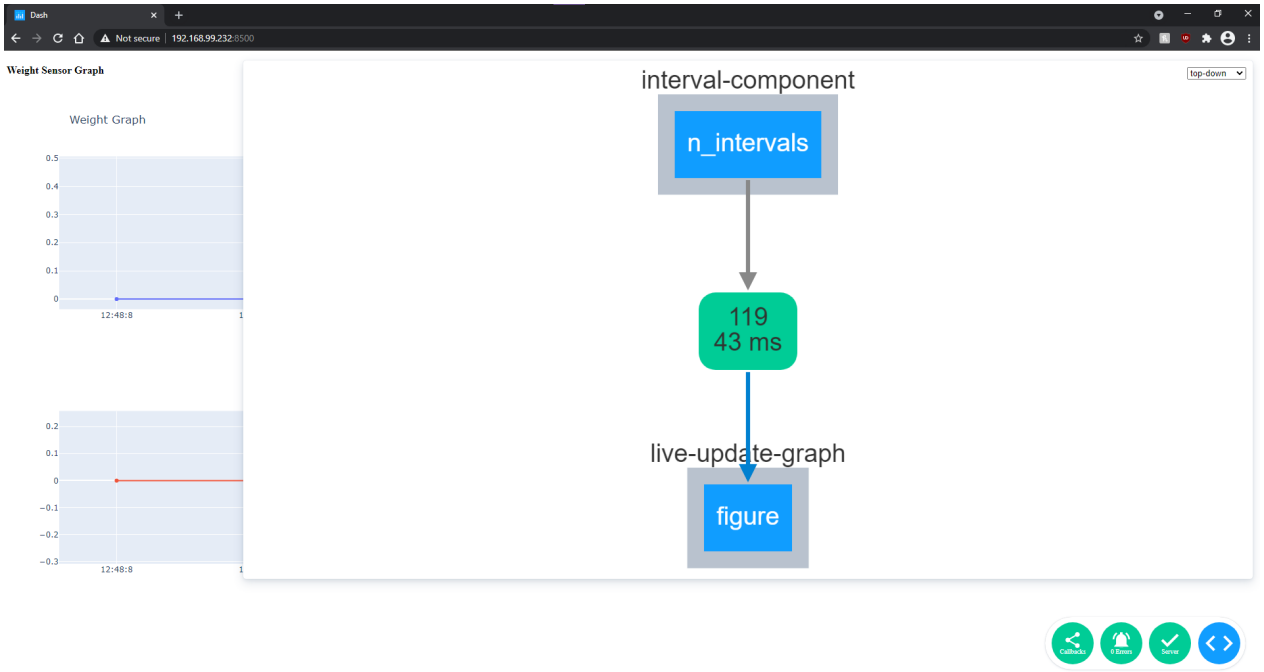
```
if __name__ == '__main__':
    app.run_server(debug=True, host="192.168.99.232", port=8500)
```

Listing 4.5: Spouštěcí funkce aplikace s parametry



Obrázek 4.5: Aplikace na lokální síti

Debugovací nástroje umožňují například zobrazit průběh a počet zavolání jednotlivých callbacků Dash aplikace. Dále zobrazuje chybové hlášení či aktuální stav serveru.



Obrázek 4.6: Sledování volání callbacku

4.1.8 Vizuální úprava aplikace

Vizuální styl aplikace byl upraven s využitím úprav stylů HTML elementů. Také je možné použít kaskádových stylů CSS. V adresáři s kódem aplikace byl vytvořen adresář s názvem **assets** a zde přidán CSS soubor s upraveným stylem. Aplikace automaticky použije CSS soubor, který se v adresáři nachází.

```
body {
background-color: #00102e;
margin: 20px;
}
```

Listing 4.6: Příklad CSS stylu

Vzhled samotných grafů byl upraven pomocí jeho zadaných vlastností při vytváření. Například nastavením parametru **line** byla upravena barva a šířka vykreslených čar, nebo parametru **marker**, upravující barvu a velikost jednotlivých bodů.

4.2 Webový server na Raspberry Pi

4.2.1 Operační systém

Pro práci s počítačem Raspberry Pi bylo nejprve nutné zvolit a nainstalovat vhodný operační systém. Jako nejvhodnější byl vybrán operační systém Raspberry Pi OS, dříve známý jako Raspbian. Systém byl nainstalován na mikro SD kartu pomocí softwaru Raspberry Pi Imager, dostupného na oficiálních stránkách Raspberry Pi. Software mikro SD kartu formátuje a následně nainstaluje vybraný operační systém.

Raspberry Pi OS je vydáván ve třech verzích, kromě normální existuje verze Lite, která neobsahuje desktopové prostředí. Poslední je verze Full, obsahující plné funkce normální verze, a navíc je zde předinstalován různý produktivní software. Byla zvolena a nainstalována normální verze systému. K počítači byl připojen monitor, klávesnice a myš. Jelikož původní model B+ nedisponuje integrovaným Wi-Fi modulem, byl pro snadnější dostupnost k internetu do volného USB konektoru připojen Wi-Fi adaptér.

Vzhledem k velikosti konektorů a Wi-Fi adaptéru bylo složité připojit do počítače i desku Arduino s připojeným senzorem. Z toho důvodu byl k Raspberry Pi připojen USB hub, jehož používání se však projevilo jako neideální. Po připojení všech periferií se objevilo varování o nízkém pracovním napětí. Jako řešení bylo na Raspberry Pi povoleno připojení pomocí SSH a do USB konektorů byl připojen pouze Wi-Fi adaptér a deska Arduino. Další práce na počítači byla prováděna pomocí softwaru PuTTY a SSH připojení z jiného zařízení.

4.2.2 Test aplikace pro tvoření grafů na Raspberry Pi

Po instalaci knihoven jako Plotly a Dash je možné spustit i hlavní aplikaci pro tvorbu grafů použitím příkazu **python**. Jelikož je ale aplikace nyní na jiném zařízení, byla editorem pozměněna nastavená IP adresa ve spouštěcí funkci kódu na lokální adresu běžícího Raspberry Pi.

Po spuštění programu byla aplikace opět dostupná na lokální síti. Po otevření adresy v internetovém prohlížeči se stránka načetla, komponenty s grafy se však nevykreslovali, a byl zobrazen pouze zbylé HTML elementy.

Pro vyřešení toho problému bylo nutné pozměnit nastavený interval pro callback aktualizace běžící aplikace. Zvýšením hodnoty **interval** v komponentu Interval z hodnoty 1000 na hodnotu 10000 se nyní aplikace bude aktualizovat každých deset sekund, na rozdíl od aktualizace každou sekundu.

```
app.layout = html.Div(
    [
        html.H4('Weight Sensor Graph'),
        dcc.Graph(id='live-update-graph'),
        dcc.Interval(
            id='interval-component',
            interval=10000,
            n_intervals=0
        ),
    ]
)
```

Listing 4.7: Úprava layoutu aplikace

Po této úpravě a opětovnému spuštění programu se komponenty grafů začali vykreslovat a aplikace běžící na zařízení Raspberry Pi byla dostupná na lokální síti.

4.3 Nastavení WSGI serveru Gunicorn

Po instalaci služby byly pro spuštění aplikace pomocí Gunicorn serveru vytvořeny dva soubory, které bude server využívat.

4.3.1 Gunicorn socket

Prvním souborem je takzvaný **socket**. Socket soubory slouží ke komunikaci jednotlivých procesů v UNIXovém prostředí. Tento socket je využit k nasměrování reverzní proxy na webovou aplikaci běžící na Gunicorn serveru. Soubor s názvem **gunicorn.socket** byl vytvořen editorem **nano** v lokaci /etc/systemd/system/gunicorn.socket.

```
[Unit]
Description=gunicorn socket
[Socket]
ListenStream=/run/gunicorn.sock
[Install]
WantedBy=sockets.target
```

Listing 4.8: Soubor gunicorn.socket

4.3.2 Gunicorn service

Druhým souborem je definice běžící služby. Soubor byl editorem vytvořen ve stejné lokaci /etc/systemd/system/gunicorn.service. Zde bylo definováno prostředí Gunicornu, pracovní adresář pro aplikaci a dříve vytvořený socket soubor. Dále byl zadán uživatel a skupina **www-group**, kterou využívají webové služby. Soubor obsahuje i příkaz a parametry ke spuštění samotné aplikace v nastavení ***ExecStart***.

```
[Unit]
Description=gunicorn daemon
Requires=gunicorn.socket
After=network.target

[Service]
User=pi
Group=www-data
WorkingDirectory=/home/pi/Desktop/pyWeightGraph
Environment="PATH=/usr/local/bin/gunicorn"


ExecStart=/usr/local/bin/gunicorn --workers 3 --chdir /home/pi/Desktop/
    pyWeightGraph --bind unix:/run/gunicorn.sock liveplotter:server --timeout 240

[Install]
WantedBy=multi-user.target
```

Listing 4.9: Soubor gunicorn.service

Služba je spuštěna se třemi pracovníky definovaných parametrem **workers**. Worker je proces, starající se o zpracování dotazů na server a odpovídání zpět klientům. Parametr **chdir** změní pracovní adresář a **bind** nastaví adresu služby na připravený **socket**. Samotná aplikace je definována názvem souboru **liveplotter** a serverového komponentu **server** definovaného v kódu aplikace. Nastavená konfigurace služby Gunicorn je spuštěna příkazem **sudo service gunicorn restart**.

```
app = dash.Dash(__name__)
server = app.server
```

Listing 4.10: Definice proměnné server

Při testování aplikace docházelo při otevření adresy v prohlížeči k chybě **504 Gateway Timeout**. Příkazem **sudo systemctl status gunicorn** bylo rovněž možné problém pozorovat, kde u každého pracovníka byla hlášena chyba **[CRITICAL] WORKER TIMEOUT**.

K tomu dochází proto, že pracovník nedokáže na dotaz odpovědět v rámci stanoveného času. Tento čas může být nastaven parametrem **timeout**. Po nastavení parametru na hodnotu 240 (sekund) je přepsáno základní nastavení (30 sekund) a po restartování služby gunicorn k chybě nedochází.

4.4 Konfigurace webového serveru Nginx

4.4.1 Definice serverového bloku

Server byl nainstalován příkazem **sudo apt-get install nginx**.

Po instalaci byl v lokaci /etc/nginx/sites-available/gunicorn.conf definován serverový blok, který definuje port a lokaci webové aplikace.

```
server {
    listen 3000;
    server_name 192.168.99.226 93.185.57.238;

    location /pyWeightGraph/ {
        root /home/pi/Desktop;
    }
    location / {
        proxy_pass http://unix:/run/gunicorn.sock;
    }
}
```

Listing 4.11: Serverový blok

V souboru byl definován pro aplikaci port 3000. Pokynem **server_name** byla zadána lokální adresa zařízení Raspberry Pi a veřejná adresa sítě, kde server běží. Blok **location** zase definuje, kde má server aplikaci hledat a směrovat dotaz. Definicí **proxy_pass**, kde byl zadán připravený socket je dotaz přesměrován na WSGI server Gunicorn.

Dále byl v adresáři /etc/nginx/sites-enabled vytvořen symbolický odkaz na tuto konfiguraci serverového bloku příkazem **sudo ln -s /etc/nginx/sites-available/gunicorn.conf /etc/nginx/sites-enabled**. Tímto je webová stránka aplikace zpřístupněna.

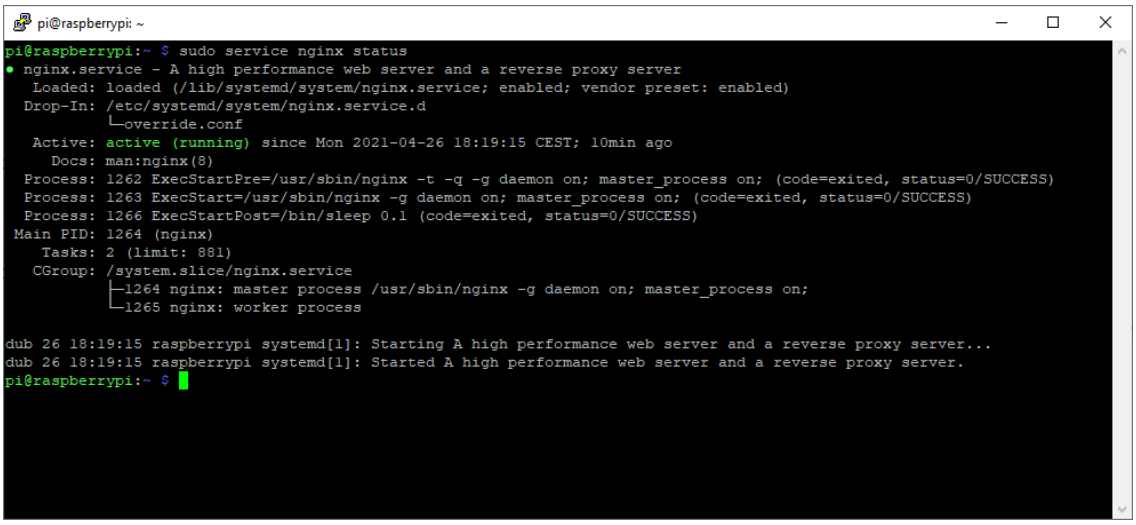
4.4.2 Konfigurační soubor Nginx

Do **http** bloku původního konfiguračního souboru byl přidán nový řádek. Přidáním řádku **include /etc/nginx/sites-enabled/*.conf;** je http blok rozšířen o dříve definovanou konfiguraci serverového bloku nacházející se v adresáři sites-enabled.

Uložením konfigurace a restartováním služby Nginx je server spušťen a aplikace je opět dostupná na lokální síti pod nastavenou adresou serveru a portem 3000.

```
user www-data;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}

http {
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
                    '$status $body_bytes_sent "$http_referer" '
                    '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile            on;
    tcp_nopush          on;
    tcp_nodelay         on;
    keepalive_timeout 65;
    types_hash_max_size 2048;

    include             /etc/nginx/mime.types;
    default_type        application/octet-stream;


    include /etc/nginx/sites-enabled/*.conf;
}
```

Listing 4.12: Konfigurační soubor nginx.conf



Obrázek 4.7: Spuštěný NGINX server

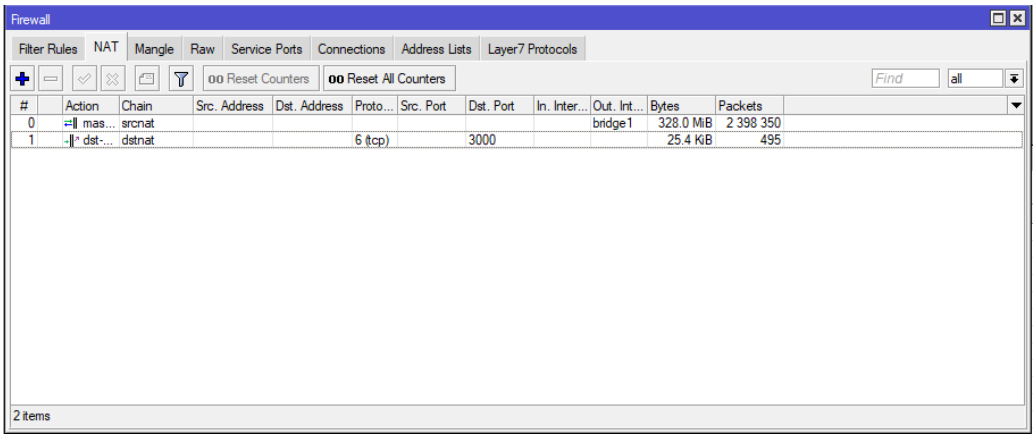
4.5 Přesměrování portu a nastavení firewallu pro přístup z internetu

Aby byla aplikace přístupná i mimo lokální síť, je nutné nastavit přesměrování portu. To umožnuje klientům z internetu posílat dotazy na Raspberry Pi server nacházející se v lokální síti.

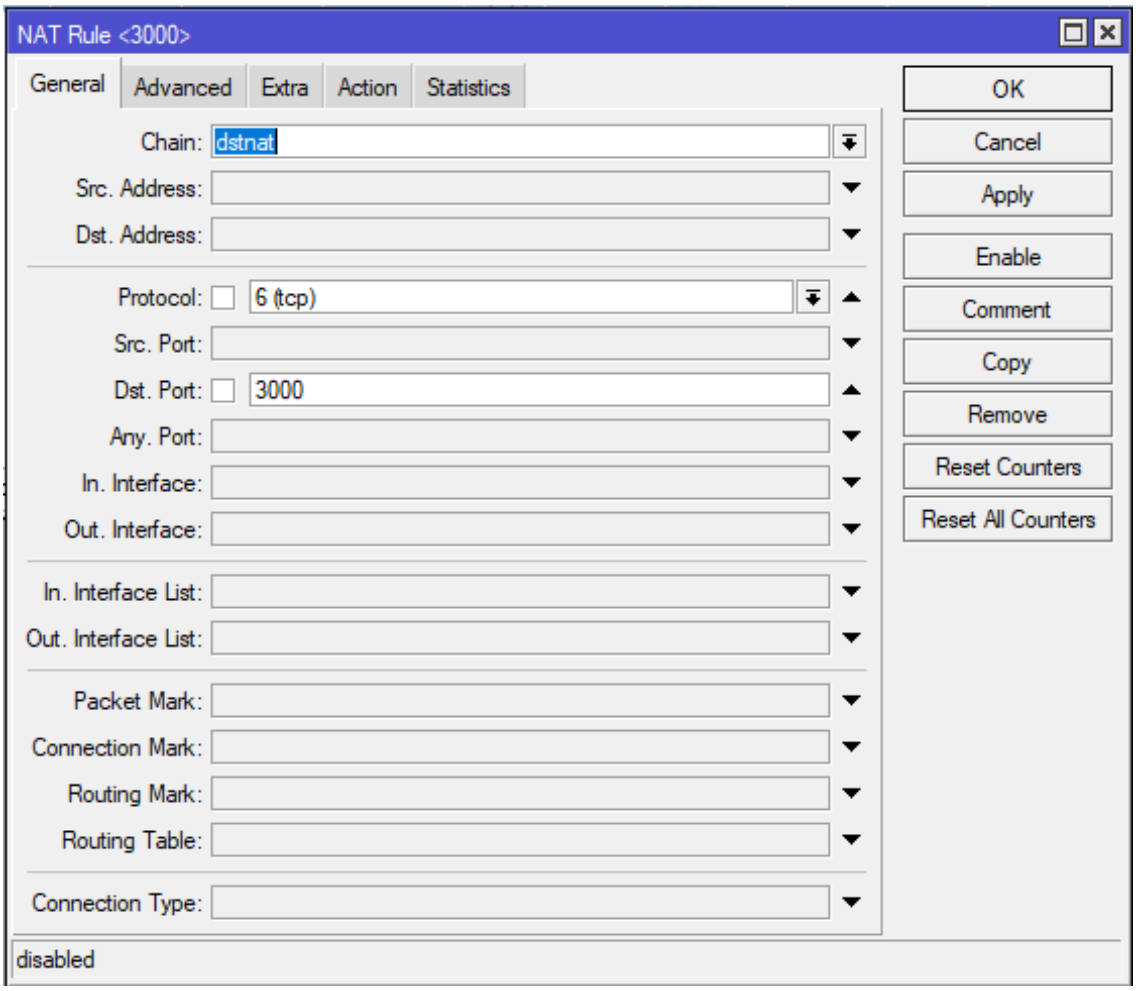
Přesměrování portu se nastavuje na hlavním routeru lokální sítě, v tomto případě je to router značky Mikrotik. Router byl konfigurován pomocí grafického rozhraní WinBox.

4.5.1 Přesměrování portu

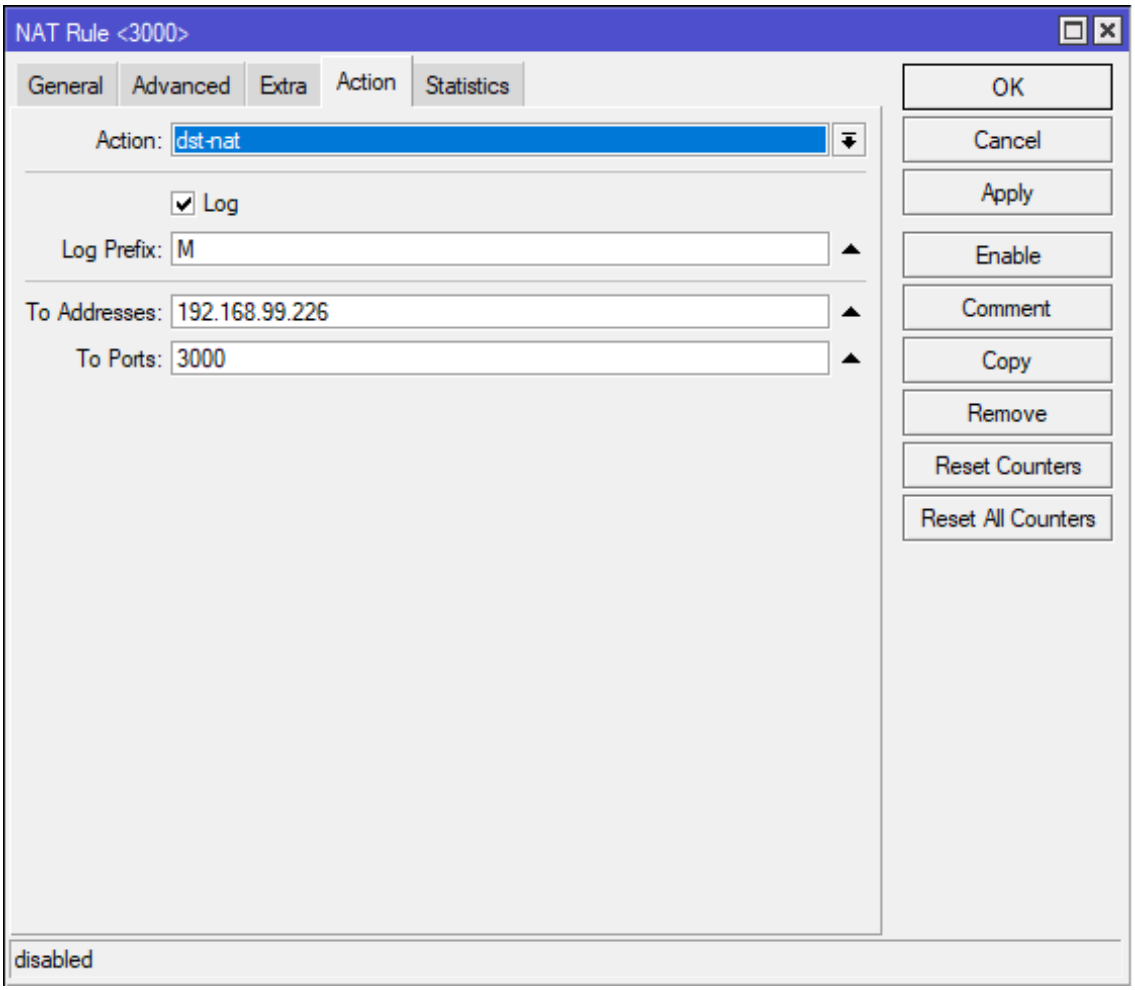
Po připojení na router pomocí rozhraní WinBox se navigací „IP → Firewall → NAT“ otevře nastavení NAT pravidel. Zde bylo vytvořeno nové pravidlo směrující TCP dotazy na port 3000 přicházející na router na lokální adresu serveru s webovou aplikací s portem 3000.



Obrázek 4.8: Winbox - Firewall NAT



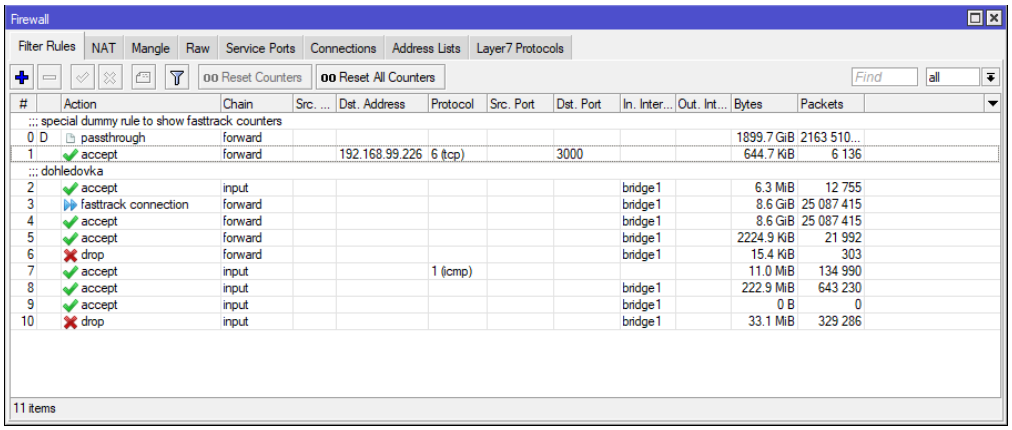
Obrázek 4.9: NAT Pravidlo - General



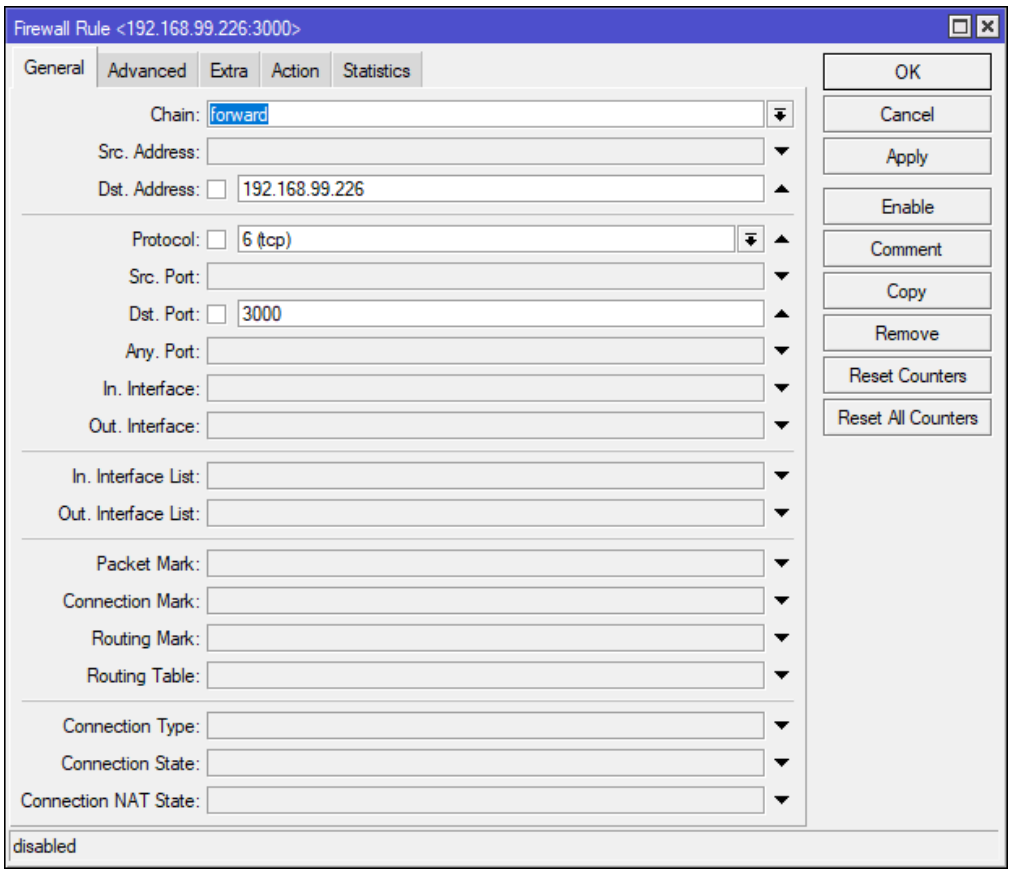
Obrázek 4.10: NAT Pravidlo - Action

4.5.2 Firewall filtr

Dále je potřeba nastavit nové pravidlo ve firewall tak, aby dotazy směřující na server Raspberry Pi byly propuštěny dále. Navigací „IP → Firewall → Filter Rules“ je otevřen seznam pravidel firewall filtru. Zde bylo vytvořeno nové pravidlo **accept**, které propustí TCP dotazy s portem 3000 namířené na IP adresu Raspberry Pi serveru. Aktivováním těchto pravidel je možné připojit se k běžící webové aplikaci i z internetu zadáním veřejné IP adresy sítě kde server běží a nastaveného portu 3000.

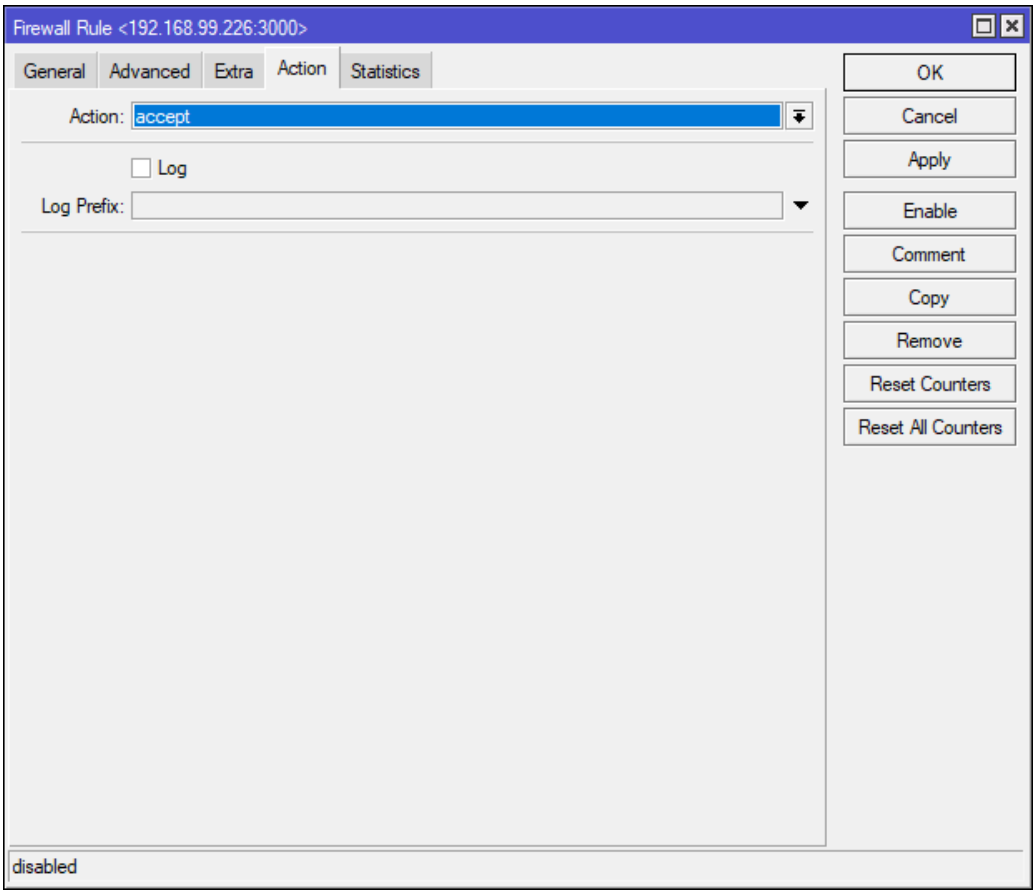


(a) Firewall Rules



(b) Firewall - General

Obrázek 4.11: Winbox - Firewall

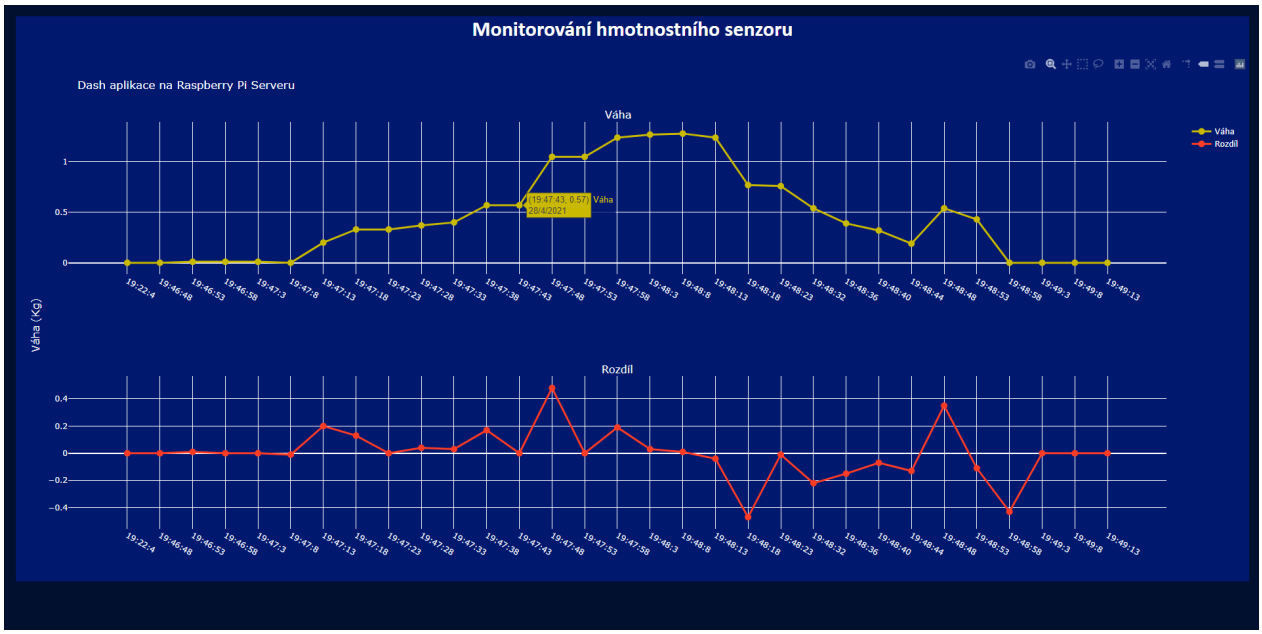


Obrázek 4.12: Firewall Pravidlo - Action

4.6 Test použití celého řešení

Webová aplikace je spuštěna zapnutím počítače Raspberry Pi a spuštěním služeb Gunicorn a NGINX. Je přístupná na lokální síti a s nastaveným přesměrováním portu i z internetu. Příkazem **python3 capture_to_db.py** byl spuštěn zapisovací program a bylo možné sledovat nové záznamy na webové stránce aplikace v reálném čase.

Při testu byly na senzor přidávány a odebírány různé potravinové produkty s rozdílnou hmotností simulující například obchodní váhu. S každým přidáním či odebráním senzor zaznamenal hmotnost produktu a ta byla zakreslena do grafu.



Obrázek 4.13: Testování běžící aplikace

Kapitola 5

Závěr

Cílem této bakalářské práce bylo navrhnout a realizovat automatizované měření hmotnosti váhovým senzorem. Získaná data poté vykreslovat do grafů na webových stránkách.

V práci bylo nejdříve popsáno, s jakými hlavními prostředky se dále pracuje. Byl zde popsán použitý hmotnostní senzor a také princip, jakým funguje. Teorie se věnuje i popisu modulu HX711, vývojové desky Arduino a počítače Raspberry Pi. Dále bylo prozkoumáno, jakým zpusobem modul HX711 a vývojová deska Arduino komunikují a přenáší data senzoru.

Prvním řešeným úkolem bylo zapojení senzoru a získání měřených dat. I přes některé problémy s použitým hardwarem bylo získání dat úspěšné a bylo možné je pozorovat na sériovém monitoru.

Dalším úkolem bylo zařídit ukládání dat do databáze. Bylo vyzkoušeno několik způsobů, ale výsledné řešení bylo provedeno napsáním programu v jazyku Python a zapisování do tabulky databázového souboru SQLite. Průběh a použitý program byly v kapitole popsány.

Čtvrtá kapitola se věnuje tvorbě webové aplikace pro vykreslení grafů a zprovoznění webového serveru na Raspberry Pi. Pro webovou aplikaci byl použit programovací jazyk Python a knihovny Plotly a Dash. Kapitola popisuje tvorbu aplikace a její postupnou úpravu. Následně ukazuje postupné nastavení webového serveru, na kterém aplikace běží. Aplikaci a server se podařilo úspěšně zprovoznit na lokální síti a po nastavení přesměrování portu i pro přístup z internetu.

Práce pro mě byla zajímavá a rád jsem si vyzkoušel práci jak s použitým hardwarem, tak softwarem. Jako největší přínos této práce bych hodnotil to, že s využitím knihovny Dash a počítače Raspberry Pi je možné vytvořit server a webovou aplikaci pro vizualizaci libovolných dat, jednoduše přístupnou z mnoha zařízení.

Literatura

1. ZEMIC. *Various types of load cells* [online] [cit. 2021-04-27]. Dostupné z: https://www.zemiceurope.com/loadcell.
2. TRENT, Dara. *Strain Gauge Load Cell Basics* [online] [cit. 2021-04-27]. Dostupné z: https://www.800loadcel.com/load-cell-and-strain-gauge-basics.html.
3. *Tenzometr* [online] [cit. 2021-04-21]. Dostupné z: https://commons.wikimedia.org/wiki/File:Strain_gauge.svg.
4. *Wheatstonův Můstek* [online] [cit. 2021-04-27]. Dostupné z: https://en.wikipedia.org/wiki/File:Wheatstone_bridge.jpg.
5. *Princip tenzometrického senzoru* [online] [cit. 2021-04-27]. Dostupné z: https://www.800loadcel.com/assets/image-cache/images/supporting-images/white%5C%20paper%5C%20images/strain%5C%20gauge%5C%202.5de4ac1b.jpg.
6. TRENT, Dara. *Types of Strain Gauges* [online] [cit. 2021-04-27]. Dostupné z: https://www.800loadcel.com/blog/types-of-strain-gauges.html.
7. ŠOFER, Michal. *Tenzometrie* [online] [cit. 2021-04-27]. Dostupné z: http://michalsofer.cz/files/EMvM/p7.pdf.
8. VOJÁČEK, Antonín. *Polovodičové tenzometry* [online]. 2006 [cit. 2021-04-27]. Dostupné z: https://automatizace.hw.cz/clanek/2006111601.
9. CWMELECTRONICA. *CZL601* [online] [cit. 2021-04-27]. Dostupné z: http://www.cwmelettronica.com/wp-content/uploads/2016/09/CZL601-Brochure.pdf.
10. *CZL601 Senzor* [online] [cit. 2021-04-21]. Dostupné z: https://dratek.cz/arduino/1209-hmotnostni-senzor-100-kg-pro-kuchynske-vahy.html.
11. SEMICONDUCTOR, AVIA. *HX711* [online] [cit. 2021-04-27]. Dostupné z: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf.
12. *Modul HX711* [online] [cit. 2021-04-27]. Dostupné z: https://ram-e-shop.com/wp-content/uploads/2018/09/kit_hx711.jpg.

13. *Arduino UNO* [online] [cit. 2021-04-27]. Dostupné z: https://store.arduino.cc/arduino-uno-rev3.

14. ARDUINO. *Arduino Products* [online] [cit. 2021-04-27]. Dostupné z: https://www.arduino.cc/en/main/products.

15. FOUNDATION, RASPBERRY PI. *Raspberry Pi Products* [online] [cit. 2021-04-27]. Dostupné z: https://www.raspberrypi.org/products/.

16. *Raspberry Pi B+* [online] [cit. 2021-04-27]. Dostupné z: https://rpishop.cz/1847-large_default/raspberry-pi.jpg.

17. MITCHELL, Robin. *UART, SPI, and I2C* [online]. 2018 [cit. 2021-04-27]. Dostupné z: https://maker.pro/arduino/tutorial/common-communication-peripherals-on-the-arduino-uart-i2c-and-spi.

18. PEŇA, Eric; LEGASPI, Mary G. *UART: A Hardware Communication Protocol* [online]. 2020 [cit. 2021-04-27]. Dostupné z: https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html#.

19. PLOTLY. *Dash Python User Guide* [online] [cit. 2021-04-27]. Dostupné z: https://dash.plotly.com/.

20. *WIND SPEED STREAMING* [online] [cit. 2021-04-27]. Dostupné z: https://dash-gallery.plotly.host/dash-wind-streaming/.

21. *Gunicorn* [online] [cit. 2021-04-27]. Dostupné z: https://gunicorn.org/.

22. *Realterm: Serial Terminal* [online] [cit. 2021-04-27]. Dostupné z: https://realterm.sourceforge.io/.

23. *NGINX Documentation* [online] [cit. 2021-04-27]. Dostupné z: https://docs.nginx.com/.

24. *SQLite* [online] [cit. 2021-04-27]. Dostupné z: https://www.sqlite.org/index.html.

25. BOGDE. *HX711* [online] [cit. 2021-04-27]. Dostupné z: https://github.com/bogde/HX711.

26. DIAGRAMS.NET. *diagrams.net* [online] [cit. 2021-04-27]. Dostupné z: https://www.diagrams.net/.

27. DEGRAWST. *Arduino Scale With 5kg Load Cell and HX711 Amplifier* [online] [cit. 2021-04-27]. Dostupné z: https://www.instructables.com/Arduino-Scale-With-5kg-Load-Cell-and-HX711-Amplifi/.

28. PLOTLY. *Python HTML Reports in Python/v3* [online] [cit. 2021-04-27]. Dostupné z: https://plotly.com/python/v3/html-reports/.

29. MARKOBIGDATA. *Nginx, Gunicorn and Dash on CentOS* [online]. 2019 [cit. 2021-04-27]. Dostupné z: https://markobigdata.com/2019/12/05/nginx-gunicorn-and-dash-on-centos/.

Příloha A

Zdrojový kód pro Arduino

```
#include "HX711.h"

#define DOUT 9
#define CLK 8

HX711 scale(DOUT, CLK);

float calibration_factor = -11400; // -11400
float data = 0;


//
    ================================================================================================

//                        SETUP
//
    ================================================================================================

void setup() {
  Serial.begin(9600);

  scale.set_gain(32);
  scale.power_up();

  scale.set_scale();
  scale.tare(); //Reset cteni na 0
```

```
}

//
    ================================================================================================

//                        LOOP
//
    ================================================================================================

void loop() {

  scale.set_scale(calibration_factor); //Aplikace kalibracniho faktoru

  Serial.println(data);

  if(Serial.available())
  {
    char temp = Serial.read();
    if(temp == '+' || temp == 'a')
      calibration_factor += 10;
    else if(temp == '-' || temp == 'z')
      calibration_factor -= 10;
    else if(temp == 's')
      calibration_factor += 100;
    else if(temp == 'x')
      calibration_factor -= 100;
    else if(temp == 'd')
      calibration_factor += 1000;
    else if(temp == 'c')
      calibration_factor -= 1000;
    else if(temp == 'f')
      calibration_factor += 10000;
    else if(temp == 'v')
      calibration_factor -= 10000;
    else if(temp == 't')
      scale.tare(); //Reset vahy na 0
  }
```

```
  delay(5000);
}
//
    ================================================================================================
```

Listing A.1: Zdrojový kód pro čtení hodnot senzoru

Příloha B

Kód pro zápis dat do databáze

```
import serial
import sqlite3
import datetime
import time

ser = serial.Serial('COM6', 9600)
ser.flushInput()
print('Serial Port is open : ' + ser.name)


def read_serial():
    line = ser.readline().decode('ascii').rstrip()
    print("read")
    return line


def db_write(time, w_data, d_data, date):
    connection = sqlite3.connect('weightData.db')
    c = connection.cursor()

    c.execute(""" CREATE TABLE IF NOT EXISTS weightdata (time TEXT, weight TEXT,
        difference TEXT, date TEXT)""")
    c.execute(""" INSERT INTO weightdata (time, weight, difference, date)
    VALUES (?, ?, ?, ?)""", (time, w_data, d_data, date))

    connection.commit()
```

```
    c.close()
    connection.close()


last_data = 0.00
while True:
    w_data = read_serial()
    if w_data is not None:
        d_data = float(w_data) - float(last_data)
        d_data = round(d_data, 3)
        last_data = round(float(w_data), 3)
        timer = datetime.datetime.now()
        t_data = '{}:{}:{}'.format(timer.hour, timer.minute, timer.second)
        date = '{}/{}/{}'.format(timer.day, timer.month, timer.year)

        print(t_data, str(w_data), str(d_data), str(date))
        db_write(t_data, str(w_data), str(d_data), str(date))
    else:
        print("NONETYPE")

    time.sleep(4)
```

Listing B.1: Kód pro zápis dat do databáze

Příloha C

Zdrojový kód webové aplikace

```
import dash
from dash.dependencies import Output, Input
import dash_core_components as dcc
import dash_html_components as html
import plotly
from plotly.subplots import make_subplots
import sqlite3


def get_db_data(db_name):
    connection = sqlite3.connect(db_name)
    c = connection.cursor()

    c.execute(""" SELECT * from weightdata """)
    dbdata = c.fetchall()

    connection.commit()
    c.close()
    connection.close()

    time_data = [x[0] for x in dbdata]
    weight_data = [x[1] for x in dbdata]
    difference_data = [x[2] for x in dbdata]
    date_data = [x[3] for x in dbdata]
    return time_data, weight_data, difference_data, date_data
```

```
colors = {
    'frame': '#00102e',
    'background': '#00186e',
    'text': '#ffffff'
}

styleback = {'align-items': 'center', 'justify-content': 'center', '
    backgroundColor': colors['frame'],
            'color': colors['frame'],}
stylemain = {'align-items': 'center', 'justify-content': 'center', '
    backgroundColor': colors['background'],
            'color': colors['text']}
styletext = {'align-items': 'center', 'justify-content': 'center', '
    backgroundColor': colors['background'],
            'color': colors['text'], 'text-align': 'center', 'font-family': '
               Calibri'}

app = dash.Dash(__name__)

server = app.server

app.layout = html.Div(style=stylemain, children=
    [
        html.H1('Monitorování hmotnostního senzoru', style=styletext),
        html.Div(id='live-update-text', style=stylemain),
        dcc.Graph(id='live-update-graph'),
        dcc.Interval(
            id='interval-component',
            interval=10000,
            n_intervals=0
        ),
    ]
)


@app.callback(Output('live-update-graph', 'figure'),
             Input('interval-component', 'n_intervals'))
```

```
def update_graph_live(n):
    data = {
        'Time': [],
        'Weight': [],
        'Difference': [],
        'Date': []
    }

    # Získání dat
    time_data, weight_data, difference_data, date_data = get_db_data('weightData.
        db')
    data['Time'].extend(time_data)
    data['Weight'].extend(weight_data)
    data['Difference'].extend(difference_data)
    data['Date'].extend(date_data)


    # Tvorba grafu
    fig = plotly.subplots.make_subplots(rows=2, cols=1, subplot_titles=("Váha", "
        Rozdíl"),
                                        y_title='Váha (Kg)')

    fig.append_trace({
        'x': data['Time'],
        'y': data['Weight'],
        'text': data['Date'],
        'name': 'Váha',
        'mode': 'lines+markers',
        'type': 'scatter',
        'line': dict(color='#c9b900', width=3),
        'marker': dict(color='#c9b900', size=10)
    }, 1, 1)
    fig.append_trace({
        'x': data['Time'],
        'y': data['Difference'],
        'text': data['Date'],
        'name': 'Rozdíl',
        'mode': 'lines+markers',
```

```
        'type': 'scatter',
        'line': dict(color='#ff3c26', width=3),
        'marker': dict(color='#ff3c26', size=10)
    }, 2, 1)

    fig.update_yaxes(type = "linear")

    fig.update_layout(
        title_text="Dash aplikace na Raspberry Pi Serveru", height=800,
        plot_bgcolor=colors['background'],
        paper_bgcolor=colors['background'],
        font_color=colors['text']
    )

    print(data)

    return fig


if __name__ == '__main__':
    app.run_server()
```

Listing C.1: Zdrojový kód webové aplikace